UC Irvine
UC Irvine Electronic Theses and Dissertations

Title
Visualization of Mobile Network Activity

Permalink
https://escholarship.org/uc/item/95p4m4c1

Author
Sathyamurthy, Nivedhita

Publication Date
2017

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

UNIVERSITY OF CALIFORNIA,
IRVINE

Visualization of Mobile Network Activity

THESIS

submitted in partial satisfaction of the requirements
for the degree of

MASTER OF SCIENCE

in Computer Engineering

by

Nivedhita Sathyamurthy

Thesis Committee:
Associate Professor Athina Markopoulou, Chair
Professor Carter T. Butts
Assistant Professor Aparna Chandramowlishwaran

2017

© 2017 Nivedhita Sathyamurthy

DEDICATION

To my parents for their unconditional love and support.

TABLE OF CONTENTS

Page

LIST OF FIGURES **v**

LIST OF TABLES **vi**

LIST OF ALGORITHMS **vii**

ACKNOWLEDGMENTS **viii**

ABSTRACT OF THE THESIS **ix**

1 Introduction **1**

2 Related Work **5**
- 2.1 Existing Applications 5
- 2.2 Visualization Approaches 6
  - 2.2.1 Data Collection and Mining 6
  - 2.2.2 Techniques in Rendering 7
- 2.3 AntMonitor 8

3 Visualization on the Mobile Device **10**
- 3.1 Overview 10
- 3.2 Visualization objectives 11
- 3.3 System Design 12
  - 3.3.1 Offline Visualization 12
  - 3.3.2 Online Visualization 13
- 3.4 Graphical User Interface 19
- 3.5 Performance Optimization 20
- 3.6 Example Visualizations 21
- 3.7 Performance Evaluation 21

4 Visualizations on the Web **23**
- 4.1 Overview 23
- 4.2 Data Collection and Parsing 24
- 4.3 System Design 25
- 4.4 Performance Optimization 28
- 4.5 Example Visualizations 29

4.6 Performance Evaluation 29

5 Conclusion **31**

Bibliography **33**

LIST OF FIGURES

Page

1.1 Permissions requested by `Flashlight` application 2
1.2 Sample visualization from `LightBeam`. The circles represent the web page user intended to visit and the triangles represent the third party sites visited by the user in real-time. Visualization similar to this did not exist for network activity by mobile applications prior to this thesis. 3

3.1 `AntMonitor` with offline and online visualization modules 12
3.2 Online visualization with methods numbered according to the sequence of actions. 14
3.3 Representation of the graph data structure containing the network activity data. The bipartite graph depicts the set of applications connected to the set of servers. An edge indicates an open connection between an application and a destination IP and an edge appears as long as the connection is active. . . 15
3.4 Screen shot of Online visualization on the mobile device 21
3.5 Time taken for loading initial connections 22

4.1 Server-side Visualizations with methods numbered according to sequence of actions 25
4.2 Screenshot of the `AntMonitor` Web. Users can select time period for viewing visualization 26
4.3 Representation of the graph data structure containing the network activity data. The bipartite graph contains set of applications connected to set of remote servers. Remote servers are colored depending the labeled group they belong to. 27
4.4 Visualization on the Antmonitor Website 28
4.5 Graphs showing the time taken to load the visualizations on the server for various simulated conditions 29

LIST OF TABLES

Page

5.1 Summary of Visualization Capabilities 32

LIST OF ALGORITHMS

Page

1 Pseudo-code for adding a link to the existing Network Activity Graph 16

2 Pseudo-code for adding a new network connection into the Network Activity Graph 17

3 Pseudo-code for removing connection from Network Activity Graph 18

ACKNOWLEDGMENTS

I would like to thank my advisor Dr.Athina Markopoulou for her encouragement and guidance. I could not have asked for a more patient and motivating advisor.

I would also like to thank my committee - Dr.Aparna Chandramowlishwaran and Dr.Carter T. Butts. I am especially grateful to Dr.Carter T. Butts for his valuable insights during the various phases of this project.

Next, I would like to thank my lab mates: Anastasia Shuba for helping me integrate the visualization with AntMonitor and for her continuous support and insight; Minas Gjoka for bootstrapping the AntMonitor website and also for his help and advice.

I would also like to thank my family and friends for supporting and motivating me. I owe my parents - Sathyamurthy and Radha for their continuous encouragement love and support, for this would not be possible without them. I express my gratitude to my friend, Aiyswarya, for always believing in me even when I failed to.

I am thankful to NSF grants 1228995, 1028394 and 1649372; and to Data Transparency Lab for funding the AntMonitor project. I would also like to acknowledge open source libraries - D3.js, BeautifulSoup, I used in my project.

ABSTRACT OF THE THESIS

Visualization of Mobile Network Activity

By

Nivedhita Sathyamurthy

Master of Science in Computer Engineering

University of California, Irvine, 2017

Associate Professor Athina Markopoulou, Chair

As smartphones and mobile devices are becoming ubiquitous and mobile applications increasingly use the network both while active or in the background, users are unaware of the continuous network activity on their phone. In this thesis, we use fine-grained network traffic measurements from mobile devices to make users aware of the network traffic activity on their devices. We use `AntMonitor`[19] - a system for monitoring network traffic, to intercept network packets in mobile devices. We use the data collected by `AntMonitor`, to provide visualization of network activity, not only on a server but also - and for the first time- on the device and in real-time. The visualizations on the phone informs the users about the servers the different applications on the phone communicate with over the Internet. The visualization on the server, with the data collected from the phone, gives the user a more detailed perspective of the network activity on their phones and over a period of time. In this thesis we describe the design and implementation of these visualizations and we discuss how they can be used to enhance the users' understanding about the network activity on their phones.

Chapter 1

Introduction

Mobile devices and smart phones have become ubiquitous and the number of unique cellular subscribers have reached half the population in the world [16]. More and more people are using mobile devices for Internet access that the usage has surpassed traditional desktop Internet usage [9]. With mobile devices becoming smart, our reliance and trust on them for daily activities have also been increasing. Mobile applications tend to have access to Personally Identifiable Information (PII) on the device such as *device IDs, contact information, location etc.*

Mobile applications, typically request access to various resources, including PIIs and the network during installation. However, the user does not often notice when these resources are accessed, or when PIIs are sent over the network to remote servers, in real-time. Sometimes, there are applications that have access to the network and other resources with no obvious functionality requirement. For example, the Flashlight application[6] in Google Play Store requests access to the network, contacts and storage as shown in Figures 1.1(a), 1.1(b), 1.1(c) whereas the description of the application does not state a reason for these requests. These applications, with access to the abundant private information on the phone pose a threat of

(a) `Flashlight` requesting access to location data

(b) `Flashlight` requesting access to memory

(c) `Flashlight` requesting access to Wifi network

Figure 1.1: Permissions requested by `Flashlight` application

leaking this information to servers that are malicious or act as *trackers* or *Adservers*. The users are currently unaware of the applications that use their data and the servers the data is sent to.

Ongoing work in the field of network traffic logging and identifying trackers on mobile devices include applications that require android devices to be rooted[11], browsers specifically identifying trackers[7], applications that display IPs contacted in a list format[10].

Ongoing work on visualizing data collected from applications, on the mobile and on a data server, include pseudo real-time pie-chart visualization requiring rooted device[15], web based pseudo real-time line charts of network flow[17]. Other interesting visualizations[12] require devices to be rooted.

One of the effective visualizations towards understanding the different servers contacted per HTTP request is provided by `Lightbeam`[8], an add-on for the `Firefox` web browser. Lightbeam depicts the various websites the user intends to browse and the third-party web pages that are contacted by the website in real-time. Figure 1.2 shows a sample capture by Lightbeam. To the best of our knowledge similar visualization does not exist for mobile traffic generated by apps, and this is what we set out to do in this thesis.

In this thesis, we attempt to make the mobile network activity more transparent to the average user. We build on the `AntMonitor`[19] application and we extend it for visualizing

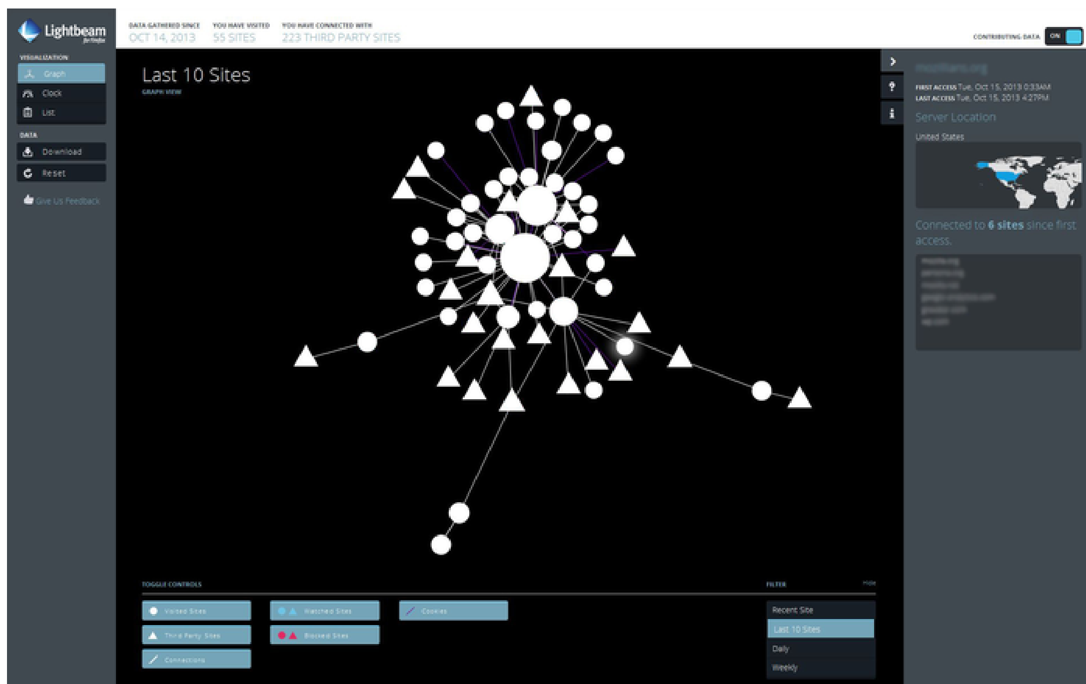


Figure 1.2: Sample visualization from `LightBeam`. The circles represent the web page user intended to visit and the triangles represent the third party sites visited by the user in real-time. Visualization similar to this did not exist for network activity by mobile applications prior to this thesis.

mobile network traffic that provides the users with an insight into the applications running on their device and the various servers they communicate with. We update the visualization on the device in real-time and we provide the users with an opportunity to be able to take a look at the historical data if they choose to upload the logs to our server. This application runs on devices without requiring administrative privileges (rooting). We make the visualizations dynamic, easy-to-understand and we provide descriptive statistics about the network activity on the phone.

To achieve this objective, we use and build on the `AntMonitor`[19] - a system for monitoring network traffic to intercept network packets. We then mine the intercepted packets for the necessary information and provide the visualizations. AntMonitor also provides means to upload the network packet logs to a remote server with the user's permission. We use these packet logs on the server to provide a historical perspective to the user through a web

application running on the server.

We develop two versions of the visualization on the phone: *Online* and *Offline*. The Offline visualization is near-real-time where we visualize the network traffic after the packet has been sent from the phone. The visualization lasts the duration of the current session, where a session is defined as the time between opening and closing of the visualization screen. The Online visualization is real-time i.e, we visualize the traffic as it is transmitted and intercepted on the phone. This visualization gives the users a sense of when the connection between an application and a server has ended as mobile applications open and close connections with remote servers continuously. Online visualization is more dynamic *i.e.* is updated as connections are created and teared-down on the phone, than the Offline visualization and the users become more aware of the current active open TCP connections on the phone.

The rest of the thesis is structured as follows. Chapter 2 describes the related work including a brief overview of the `AntMonitor`. Chapter 3 presents the visualization on the mobile device, including offline and online visualizations. Chapter 4 presents the visualizations on a web server. Chapter 5 concludes the thesis and outlines directions for future work.

Chapter 2

Related Work

2.1 Existing Applications

In this section, we take a look at some of the existing applications that provide network activity visualizations.

Network Log[11] provides real time list of applications that have an open network connection at any time instance and provides descriptive statistics. Contains a time line chart where the number of packets sent by each application (y-axis) is plotted against time (x-axis). Packet details of upto 48 hours can be visualized. This application, however violates the SELinux policy and does not run in user space (needs administrative privilege).

Ghostery[7] is a browser that specifically caters to providing users information regarding the trackers the user fell victim to.

Network Connection[10] displays a list of all the active network connections in the phone. The application displays bytes sent and IPs contacted by each app in a list format. Geographical location of an IP is also provided along with the host name.

`Lightbeam[8]` is an add-on for the `Firefox` web browser that depicts the network activity of the user on the web browser. Figure 1.2 is an example screen shot of the `Lightbeam` page on the web. There is no such real-time representation of the network activity of mobile applications. In this thesis, we set out to provide similar visualization for network activity for the mobile device.

2.2 Visualization Approaches

To visualize network activity on the phone, the following steps take place: collecting the traffic data, mining for the details of interest for visualizing and rendering the data. In this section, we describe the various techniques available for achieving each of these steps and outline our approach.

2.2.1 Data Collection and Mining

User space: Network traffic data can be collected inside or outside of the user space. Collecting data outside of the user space requires the android device to be rooted. If an application is collecting data using *logcat* or *dumpsys*, it runs outside of the user space. Applications running outside of user space, though have access to fine-grained data, is not suitable for all users. *Network Connections*[10] and *Flowoid*[15] are examples of applications that work outside of the user space whereas *Network Log*[11] and *Ghostery*[7] work in the user-space.

Data updating frequency: Since applications send network traffic while running in the background and foreground, the traffic data is being collected continuously. This can be updated in the visualization in real-time *i.e.* when the data is collected or in pseudo-real time *i.e.* updating the visualization after a few minutes of data collection. *Network Connection*[10]

displays the collected data in real-time whereas *Flowoid*[15] updates the visualization in near-real-time.

Our approach is to build an application that works in user space to collect network traffic. We use the `AntMonitor`[19] to intercept the network traffic and collect the necessary data. We describe the architecture of the `AntMonitor` system in section 2.3. In this thesis, we develop a visualization module on top of the `AntMonitor` application to give the users better insight into current network activities on their phone.

2.2.2 Techniques in Rendering

We represent the network traffic data as a sparse bipartite graph between the set of applications and the set of IPs. An edge between two nodes indicate the flow of traffic between them. Figure 3.3 gives the structure of the information we collect. This information can be visualized in many ways.

Pie Chart: *Flowoid*[15] uses pie-chart to display the applications and the servers they contacted, and also the ports that were used for the connection. These charts are not trivial to understand.

Time line charts: *ToA*[17] and *Network Log*[11] uses time line charts to display the number of packets transferred over a period of time. This gives a historical insight into the network traffic over the period of data collection.

Plainlist: *Network Connections*[10] displays the list of active connections as a plain list. The meta-data for each of the connections, like geographical location of the remote server, is displayed on selecting the particular network connection.

Force-directed graph drawing: In this thesis, we use the Force-directed graph drawing[14]

technique to represent the interaction between applications on the phone and the various remote locations. Force-directed graph drawing is an aesthetically pleasing technique: the graph is interpreted as a physical system with forces and try to minimize the energy of the system to obtain a nice drawing[13]. These algorithms are generally used for graphs with community structure and sparse graphs. This drawing technique is similar to the one that is used in Lightbeam for displaying the network traffic in the current instance of browser shown in Figure 1.2. Since our graph data structure is pretty sparse, we plan to use this aesthetically pleasing algorithm to draw the graph for our visualization.

2.3 AntMonitor

In this section, we give an overview of `AntMonitor`[19] - the platform we use for intercepting mobile network traffic. `AntMonitor` is a system for monitoring, collection and analysis of fine-grained, large-scale packet measurements from mobile devices. `AntMonitor` is designed as a VPN-based application that runs on the mobile, therefore the application can run on user-space (non-rooted). `AntMonitor` is designed for efficient use of resources on the device.

`AntMonitor` can be used for a variety of passive performance monitoring applications like real-time detection and prevention of privacy leaks from the device, passive performance measurements and application classification based on TCP/IP header features. Throughout this thesis, we show that the `AntMonitor` can also be used to increase transparency of network activity to users by means of real-time or historical visualizations that provide useful insights.

We use the `AntMonitor` Library to intercept the network packets and extract the necessary data from the packets. We add a component to the existing `AntMonitor` application to visualize the network activity. Figure 3.1 shows the modified architecture diagram including the visualization modules. We then use this data to visualize the network activity on the

device, as described in Chapter 3.

AntMonitor supports packet capture of both incoming and outgoing traffic. AntMonitor provides the users with an option to upload the captured data for analysis on the server. The packet traces are collected in PCAPNG format[4]. This format allows us to append arbitrary information alongside the packet capture. This capability provides us with the opportunity to add contextual information for every packet that can be later used for analysis. We append the application name that sent/received the packet, to the packet trace. This information helps us to provide historical visualizations of the network activity on the server.

To achieve this, we develop an interactive web application to give the users an insight of how different applications have been using the network over a period of time. We extract the data collected from the device and render the network activity for a particular device as described in Chapter 4

Chapter 3

Visualization on the Mobile Device

3.1 Overview

We provide two types of visualization on the device - *Offline*, a near-real-time visualization and *Online*, a real-time visualization. The two kinds of visualizations differ in terms of collecting the network activity data. The Graphical User Interface for both *Online* and *Offline* visualizations are the same. This means that we use the same graph drawing technique for both *Online* and *Offline* visualizations. To the best of our knowledge, this is the first application to visualize network activity on the mobile device in real-time.

Offline visualization is near-real time where the packet data is logged on the phone but visualized after the packet is transmitted. *Online visualization* is real-time where the packet data is visualized while the packet is on its way to transmission. Also, this visualization is more dynamic i.e this visualization effectively depicts the opening and closing of a connection hence giving the users an idea of the open at any instant of time.

The rest of this chapter is structured as follows. Section 3.2 outlines the design objectives

for the system. Section 3.3 provides a description for the methods used for extracting the information from the packets. Section 3.4 describes the Graphical User Interface designed for displaying the visualization. Section 3.5 outlines the steps taken to optimize the memory usage for better usage of memory. Section 3.6 shows examples of visualization on the mobile device. Section 3.7 reports performance metrics for the visualizations on the mobile.

3.2 Visualization objectives

Real-time packet interception: We would like to monitor and visualize the network traffic in real-time. More specifically, we want to be able to intercept the traffic in real-time and extract essential details pertaining to that packet without disrupting the flow of traffic or introducing noticeable delay. We use the `AntMonitor` Library[19] to intercept the traffic in real time. To achieve this, we extract the data from the packets when it is captured by the VPN server and just before it is transmitted.

Easy to understand visualization: We would like to visualize the data in an easy-to-comprehend way. In order to realize this, we use D3.js[1] to create a webview in android application that would visualize the details of the network packets such as the application that is sending/receiving it, the remote destination which has sent/received.

Real-time graph updating: Using the `AntMonitor` Library gives us the unique opportunity to provide real-time insight to the users about the network activity on their phones. To fully capitalize on this, we provide an interactive visualization that updates the force-graph every time a new connection is opened or closed and keeps track of the amount of data that has been transferred in the current transaction.

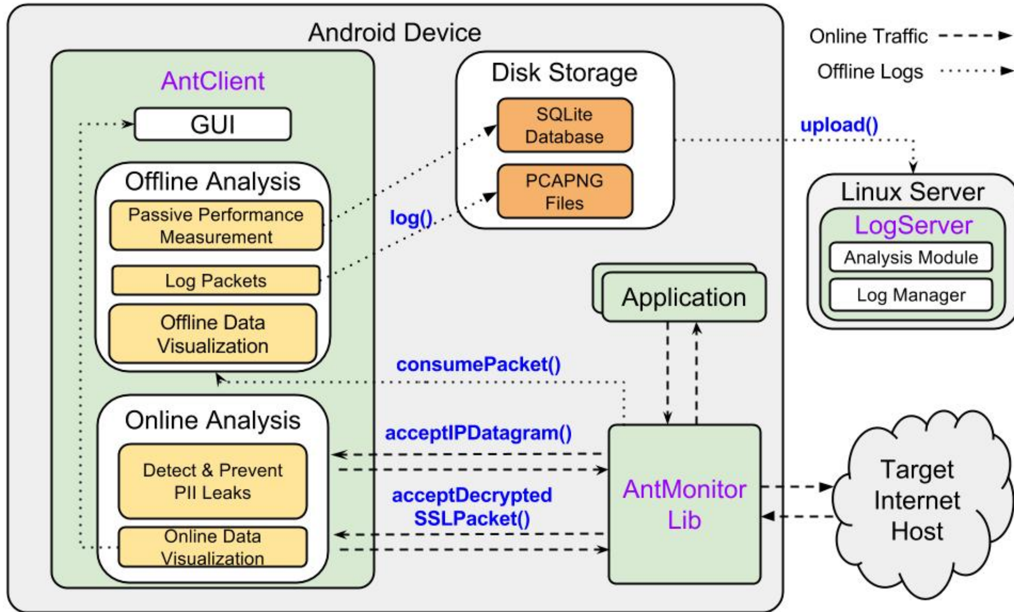


Figure 3.1: `AntMonitor` with offline and online visualization modules

3.3 System Design

We use the `AntMonitor` Library to collect the data necessary for visualizing the network traffic in real time on a mobile phone. The network packets contain details including the port the packet was sent from, the remote IP the packet was sent to, how much data was sent. These details can be collected from the network packets during or after they have been transmitted.

3.3.1 Offline Visualization

The `AntMonitor` mobile application gives an option for users to log network packet measurements to a remote server. Only the packets from applications that have been selected by the user to be logged, are sent to the remote server. Figure 3.1 shows the `AntMonitor`

architecture with the offline visualization. In this method, we intercept the packets before they are filtered to be logged. If the visualization screen is opened, we mine the stream of packets coming in to be logged. At this point, the packet is already transmitted. We mine each packet to extract the port from which it was sent, the destination IP address, the source IP address and the length of the packet. Using the `AntMonitor` Library, we also identify the package name of the application that the packet is associated with. Once we have mapped the application, we send this data to the Graphical User Interface for it to be rendered.

For every packet, the extracted data is bundled together and sent to the webview in the user interface. The webview is synchronized to process this stream of data in a sequential order. The visualization presents a bipartite graph with one set of nodes being the applications that generate traffic, and another set being the destination IP the traffic goes to. Each node also contains other meta-data like the icon associated with the application. An edge exists between two nodes if the packet is transmitted between an application and a server.

3.3.2 Online Visualization

The `AntMonitor` Library uses a VPN service to intercept network packets on the mobile device. The Forwarder module is responsible for routing network traffic on the device. For each TCP connection made by the application on the device, an instance of the TCP Forwarder is created. When the connection is closed, the instance of the forwarder is destroyed. The VPN service keeps track of the active Forwarders at any instance.

Figure 3.2 describes the visualization system. The following steps describes the series of events that trigger the visualization:

1. When the **Network Activity Screen** is opened in the `AntMonitor` application on the mobile device, **PacketFilter**.*triggerOpenConnections()* in the `AntMonitor` Library is

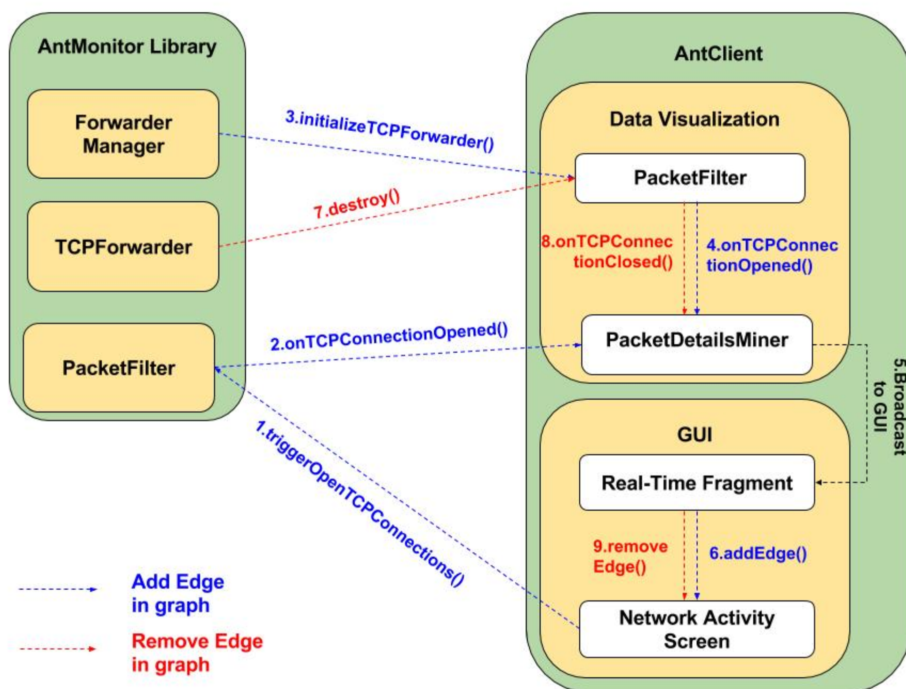


Figure 3.2: Online visualization with methods numbered according to the sequence of actions.

invoked to fetch the existing active connections. The existing active connections are mapped by the **ForwarderManager** and can be accessed by the **PacketFilter**.

2. The active open TCP Connections are retrieved and for each of the open connections, we extract the *source IP, destination IP, source port* and *destination port*. With this information, we invoke **PacketDetailsMiner**.*onTCPConnectionOpened()*, which acts as the interface between the **Data Visualization** module and the **GUI**.

3. When the `AntMonitor` Library intercepts network traffic, the **ForwarderManager** creates an instance of **TCPForwarder** for every new TCP connection on the device. **TCPForwarder**.*initializeTCPForwarder()* notifies the **PacketFilter** in the AntClient of the new TCP connection.

4. The **PacketFilter** extracts the *destination IP, source port* and *destination port* from the instance of the **TCPForwarder**. These details are then sent to **PacketDe-**

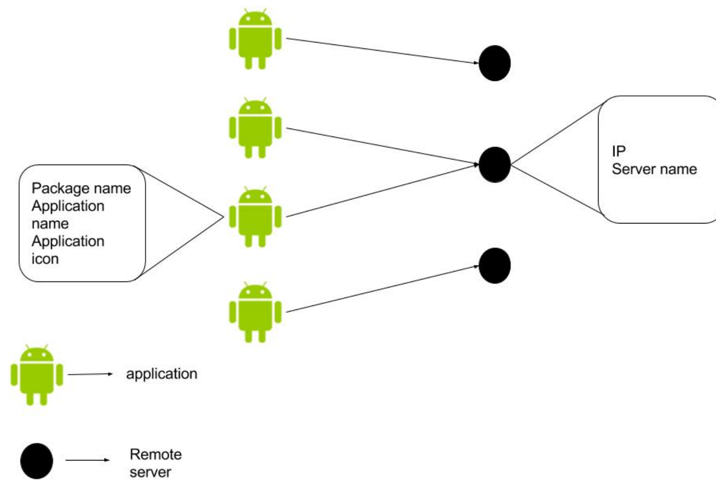


Figure 3.3: Representation of the graph data structure containing the network activity data. The bipartite graph depicts the set of applications connected to the set of servers. An edge indicates an open connection between an application and a destination IP and an edge appears as long as the connection is active.

tailsMiner.*onTCPConnectionOpened()*. Again, this method acts as the interface between the **Data Visualization** module and the **GUI**.

5. The **PacketDetailsMiner** uses the `AntMonitor` Library, *destination IP, source port* and *destination port* to identify the name of the package the packet is associated with. The *package name* and *destination IP* are then broadcast to the **Real-Time Fragment**.

Algorithm 1 Pseudo-code for adding a link to the existing Network Activity Graph

```
addLink(sourceId, targetId):
    sourceNode = findNode(sourceId)
    targetNode = findNode(targetId)
    while i < links.length do
      if links[i].source == sourceNode and links[i].target == targetNode then
            links[i].byteCount += byteCount
            return
      else
            i++
      end if
    end while
    if sourceNode not undefined and targetNode not undefined then
      links.push("source":sourceNode, "target":targetNode)
    end if
```

6. The **Real-Time Fragment** receives the *package name* and *destination IP* and uses the *package name* to fetch the icon and name associated with the package. The details - (*package name, destination IP, icon* and *length*) are then sent to the **Network Activity Screen**. The **Network Activity Screen** is a webview where the network activity is represented and rendered. The network activity is stored for the current session in the webview. The network activity is stored as a *Network Activity Graph* as represented in Figure 3.3. Algorithms 1 and 2 describe the pseudo-code for adding a new connection to the *Network Activity Graph*. To add a new edge to the graph, we check if the package and remote server already exist in the *Network Activity Graph*. If the nodes do not exist in the graph, we add the nodes to the graph. If the nodes exist in the graph, we check if there exists an edge between the package and remote server and if there is none, we add an edge between the two nodes.

Algorithm 2 Pseudo-code for adding a new network connection into the Network Activity Graph

addConnection($package - name, ip, image$**):**

  if $findNode(package - name) == undefined$ **then**

    addNode(package-name,image)

  end if

  if $findNode(ip) == undefined$ **then**

    addNode(ip)

  end if

  addLink(package-name,ip)

7. When a TCP connection is *closed*, the corresponding instance of **TCPForwarder** is destroyed. **TCPForwarder**.*destroy()* notifies the **PacketFilter** of the connection that is closed.

8. The **PacketFilter** extracts the *source IP, destination IP, source port* and *destination port* from the instance of the **TCPForwarder** that is being destroyed. These details are then sent to **PacketDetailsMiner**.*onTCPConnectionClosed()* which acts as an interface between the AntClient and the GUI. The **PacketDetailsMiner** uses the `AntMonitor` Library, *destination IP, source port* and *destination port* to identify the name of the package the packet is associated with. The *package name* and *destination IP* are then broadcast to the **Real-Time Fragment**.

Algorithm 3 Pseudo-code for removing connection from Network Activity Graph

```
removeConnection(sourceNode, targetNode):
    linkExists = false
    i=0
    for i = 0 to i < links.length do
      if links[i].source == sourceNode and links[i].target == targetNode then
            links.remove(i)
            linkExists = true
      end if
    end for
    if linkExists then
      isApp=false
      isIp=false
      for i = 0 to i < links.length do
            if links[i].source == sourceNode then
                  isApp = true
            end if
            if links[i].target == targetNode then
                  isIp = true
            end if
      end for
      if isApp == false then
            nodes.remove(sourceNode)
      end if
      if isIp == false then
            nodes.remove(targetNode)
      end if
    end if
```

9. The **Real-Time Fragment** receives the *package name* and *destination IP* from the broadcast by **PacketDetailsMiner**. The details - *package name, destination IP* are then sent to the **Network Activity Screen**. Algorithm 3 describes the pseudo-code for removing an existing connection from the *Network Activity Graph*. We check if there exists a link between the application and the remote server in the connection that is being close. If the link exists, we remove the edge from the *Network Activity Graph*. Once the edge is removed, we check if the package and remote server has other edges associated with them. If either or both of them has no edges associated with it, we remove the node(s) from the graph.

3.4 Graphical User Interface

The data mined from the stream of network packets are sent to the webview in the android application to be visualized to the users. The webview presents the stream of data into a bipartite graph. For every addition of edge, we draw the same in our visualization and similarly, for every connection that is closed, we visualize the removal of an edge.

We want to present the users with the data extracted from network packets - *AppName, Remote IP, time the connection opened, time the connection closed*. To visualize this continuous stream of data, we store this data in a dynamic graph where we add/remove nodes/edges.

Figure 3.3 shows a sample network traffic graph. We can see from the figure that it is a bipartite graph with applications on the device as one set of nodes and the remote servers being the other set. An edge exists between two nodes if there is an active TCP connection between an application and a remote server. Each node representing an application contains the *package name, application name* and *application icon* while each remote server contains *IP address of the server* and *host name*.

We use a force-directed graph to visualize the network traffic. We use the D3.js[1] implementation of the force-directed graph to realize the visualization on a web view in an Android application running on the phone. We use the force-directed graph so that all the edges in the graph are almost the same distance. Force-directed graphs are also aesthetically pleasing and present the data in a meaningful and easy-to-understand format to the users.

3.5 Performance Optimization

Data mining and rendering the visualization can be very CPU and memory intensive. As the visualization runs along with `AntMonitor`, we need to optimize the visualization in order to optimize the use of resources without draining the user's battery.

Mining data requires CPU resources to process the data and also memory to store the mined data. In order to optimize resources, we maintain the data structure to store the network traffic data in the web view, and we avoid creating any additional data structures. The web view is destroyed when the visualization screen is closed and this helps in destroying the data collected for the current session. Using a dynamic graph helps in restricting the size of the graph structure to the number of active TCP connections.

Rendering the visualization requires extensive CPU processing. The real time visualization is updated on every opening and closing of TCP connections. Since, the rate of updating is very high, the CPU requirements for this case is high too. To save the mobile device from battery drain due to excessive CPU usage, we only process the data, store it and render the visualization when the screen is on the foreground.

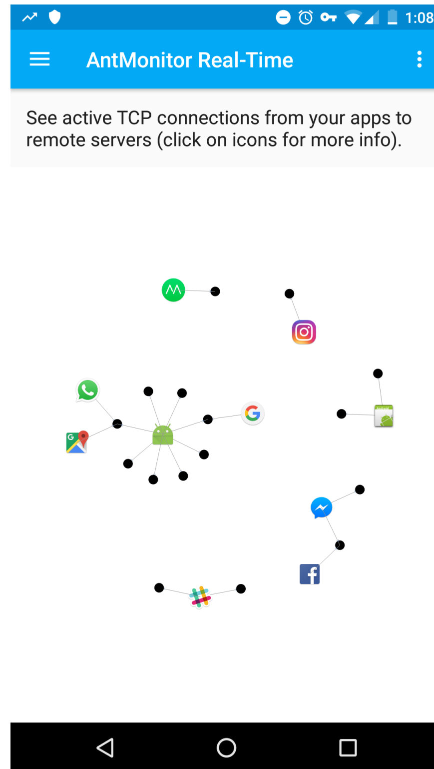


Figure 3.4: Screen shot of Online visualization on the mobile device

3.6 Example Visualizations

Figure 3.4 is a screen capture of the Online Visualization of network traffic on the mobile device of one user. The screen shot shows the various applications that have open TCP connections while this was captured. This makes the users aware of what applications are communicating over the network at the current time. This also gives users an idea of the different applications that talk to the same server.

3.7 Performance Evaluation

Setup: When the visualization screen is opened in the device, we visualize the existing open TCP connections. The existing number of connections could be as small as 1 or 2, if the

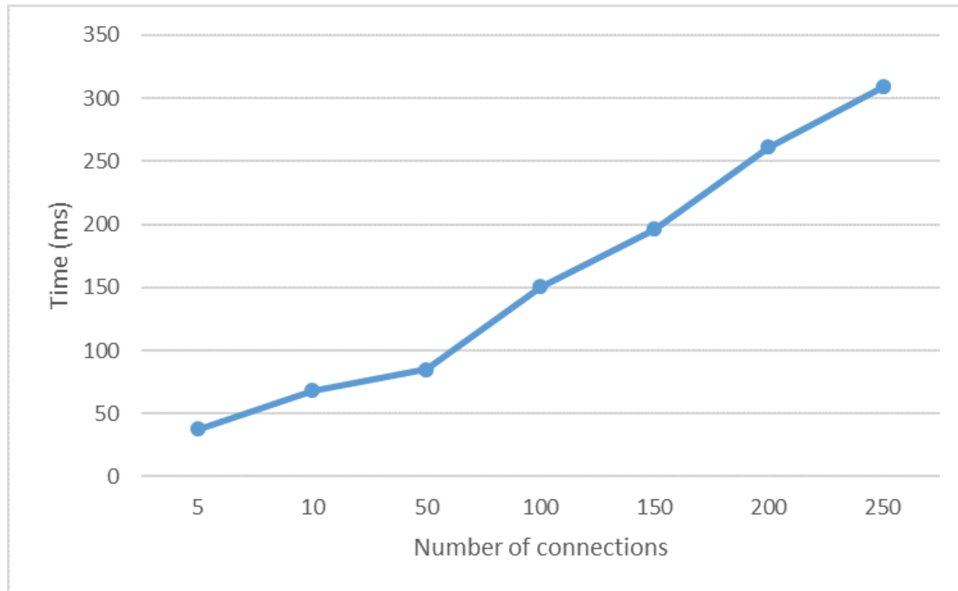


Figure 3.5: Time taken for loading initial connections

phone is not currently being used for network heavy applications, or it can be in the order of hundreds if the phone is using the network extensively. We simulate conditions similar to an idle phone (5 open connections) to busy phone (250 open connections) to evaluate how the number of connections varies the initial loading time.

Results:From Figure 3.5, we see that the time taken to render the connections increases with the number of open connections. We see that this initial wait could go up to 1 second in case of busy network activity in the phone.

This wait is only during the initial loading of the visualization screen. Once the initial open connection is rendered, we see that the visualization screen continues to draw and remove TCP connections as they are being opened and closed. Addition and removal of edges in the existing graph occurs with minimal time, averaging to about 0.1 milliseconds after the initial load.

Chapter 4

Visualizations on the Web

4.1 Overview

`AntMonitor` provides the users with an option to upload their logs to the logserver as shown in Figure 3.1. This provides us with an opportunity to visualize historical information on the server and to correlate the information collected from different users. In this thesis, we used `AntMonitor` to collect network traffic measurements from a pilot study conducted at UCI for over a year. The data is collected in the PCAPNG[4] format from the phone and is securely uploaded to a server. We want to use this data and re-create the network activity visualizations. To do this, we create a web application on the log server which the users can interface with to view the historical visualizations of their own network traffic.

The visualization on the server can be more powerful than on the mobile device and can provide visualizations with richer meta-data and can be used to identify labeled groups of servers like *Adservers, blacklists, DNS Servers*. In addition to re-creating the visualization similar to the one on the mobile, we also use the data available to identify if the servers the applications are communicating with belong to any known labeled sets. We show that this

can also be extended to any labeled sets we might want to identify.

The rest of the section is structured as follows. Section 4.2 gives an overview of the database that has the network packet measurements collected from the users of the pilot study at UCI. Section 4.3 describes the structure and implementation of the web application that visualizes the network data. Section 4.4 outlines the steps taken to improve performance of the web application. Section 4.5 shows sample visualizations on the server. Section 4.6 provides a performance evaluation of the web application rendering the visualization.

4.2 Data Collection and Parsing

`AntMonitor` allows the users to upload the network traffic measurements collected on their device to a remote server. If the user chooses to upload logs, the traffic data is collected in *pcapng* format. We have data collected from a pilot study at UCI for over a year. These traffic measurements, are then processed by the **log server** and the network packet features are extracted. From each packet, approximately 66 features are extracted which can be roughly classified as packet length statistics, payload length statistics, inter-arrival time statistics, distribution of pairs of traffic bursts, general summary statistics and TCP flags. All the above mentioned features are bundled into a tuple of the **MySQL relation** for later access.

Each tuple in the MySQL relation is uniquely identified by unique user id provided by `AntMonitor` mobile application. `AntMonitor` provides a way to map the network packets to applications on the phone. `AntMonitor` maps each packet to the application that sent it before it is converted into pcapng format. We store the `AntMonitor` User ID and the application package name along with network packet features in the MySQL relation.

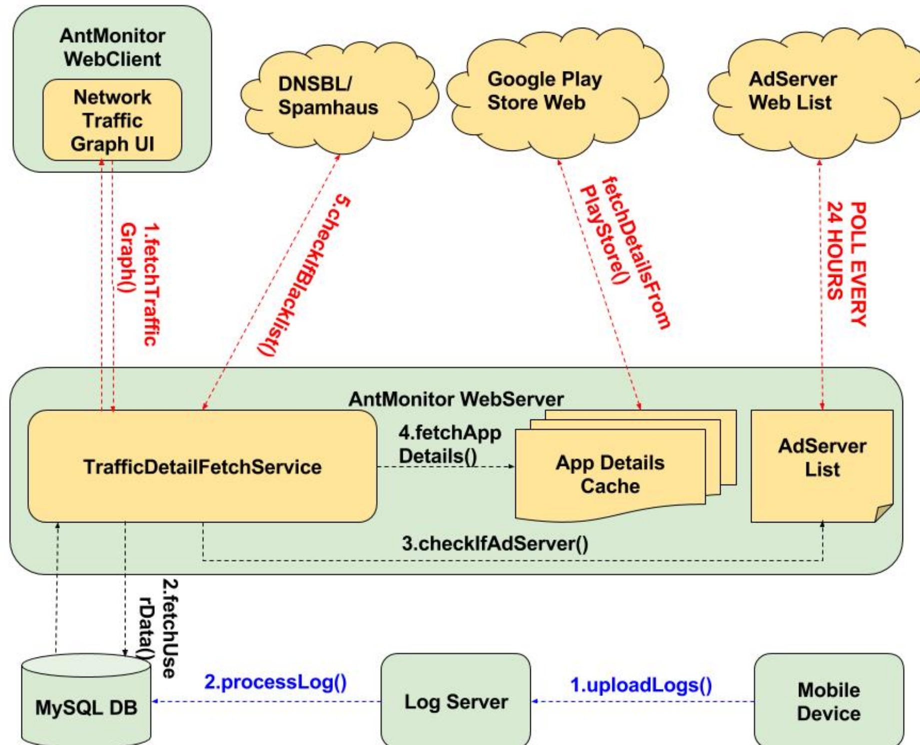


Figure 4.1: Server-side Visualizations with methods numbered according to sequence of actions

4.3 System Design

Figure 4.1 provides a block diagram representation of the `AntMonitor` website. The data collected from mobile devices are uploaded to the log server and are entered in the MySQL Database. We use this database for retrieving user data for visualization.

The following steps outlines the series of events that occur for loading the network activity visualization in the `AntMonitor` website:

1. The user logs in the website with their AntMonitor id and password and can see a screen similar to Figure 4.2. This is the base of **Network Traffic Graph UI** and it provides the users with an option to select the time period for which they want to view the network traffic activity of their device. When the users select the time period,

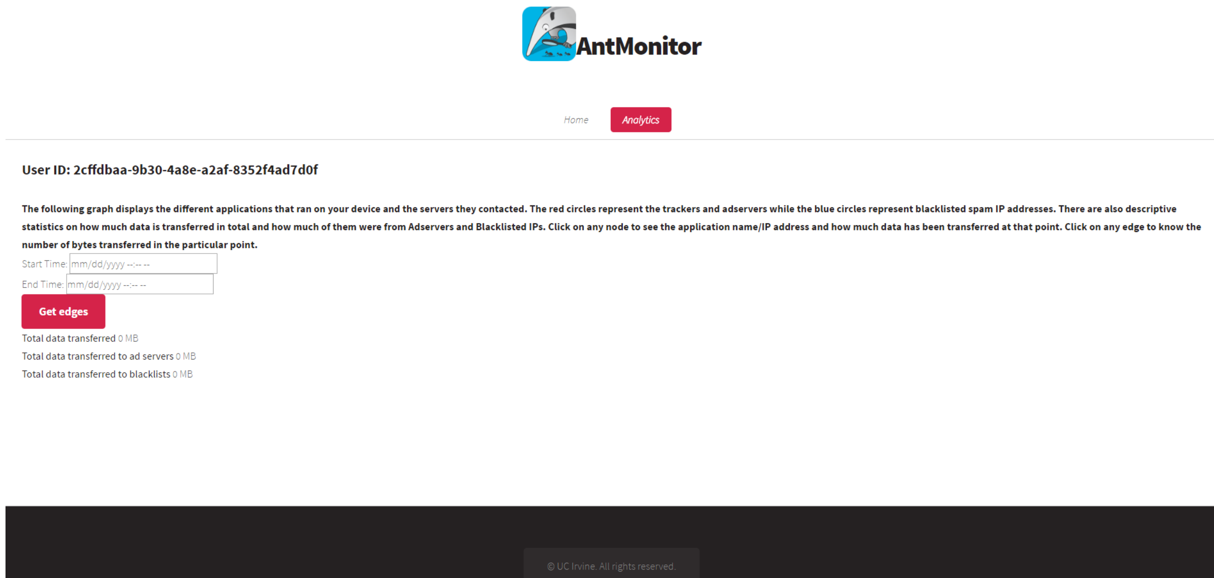


Figure 4.2: Screenshot of the `AntMonitor` Web. Users can select time period for viewing visualization

TrafficDetailFetchService.*fetchUserData()* is invoked for fetching the details for the user given the time period.

2. The **TrafficDetailFetchService** queries the MySQL database for the user data in the time period specified. The database returns a set of tuples in the chronological order. Each tuple consists of *time when first packet was sent, time when last packet was sent, total number of bytes transferred, package name* and *IP address of remote server*.

3. For every application in the result, *fetchAppDetails()* looks up for the image and developer details associated with it in the **App Details Cache**. Once we have the details of the application, we proceed to finding information regarding the remote server.

4. Once every day, we poll the web for a list of adservers[3]. We store the **AdServer List** on the **LogServer**. We identify the IP addresses on our result that are present in the **Adserver List**. We do a subnet matching against our list to increase the adserver identification. The results are then mapped for each network connection.

5. We query the DNSBL[2] and Spamhaus[5] services to check if the IP address of the

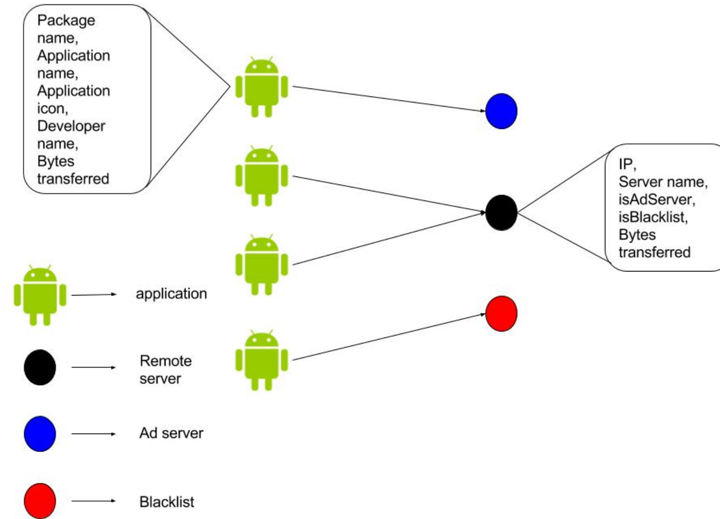


Figure 4.3: Representation of the graph data structure containing the network activity data. The bipartite graph contains set of applications connected to set of remote servers. Remote servers are colored depending the labeled group they belong to.

remote server is blacklisted. For every IP address we have in the result of our query, we identify if it is blacklisted and add it to the results.

6. Once the necessary information for creating the traffic graph is extracted, we send this data back to the **Network Traffic Graph UI** where the graph will be rendered in a force-directed format. Figure 4.3 describes the structure of a sample network traffic graph along with the meta-data of the nodes.

Suspicious overlaps: Applications tend to communicate with multiple remote servers at any point of time.Sometimes, different applications may talk to the same server(s), some of which can be ad servers, spam, or DNS servers. We indicate such remote servers that have been contacted by applications from different developers by marking them green.

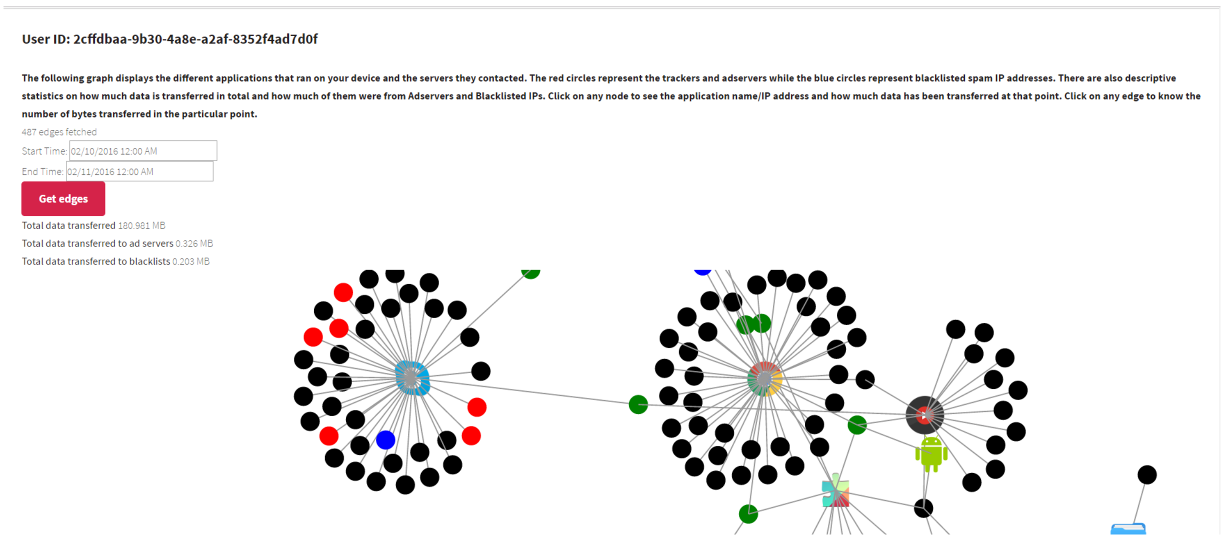
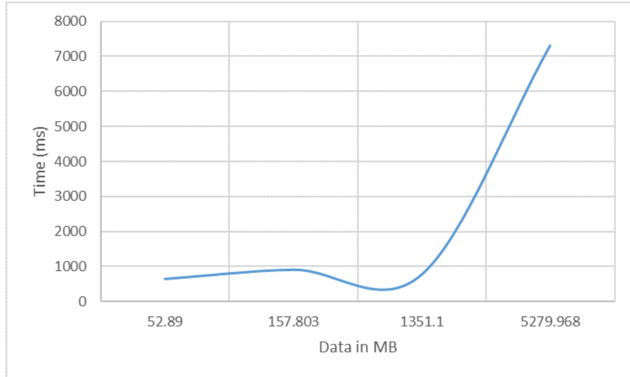


Figure 4.4: Visualization on the Antmonitor Website

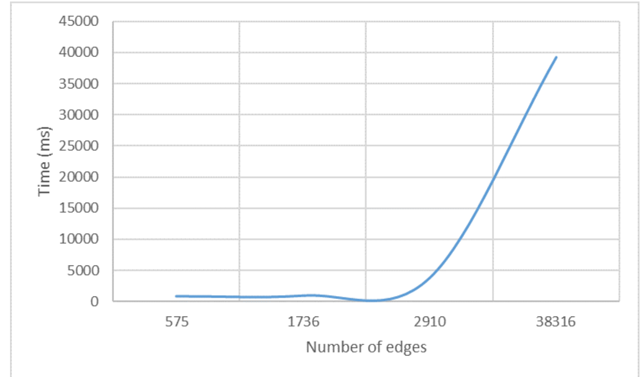
4.4 Performance Optimization

Rendering of web applications becomes challenging with multiple network calls to fetch other details. In our case, to render the visualization, we get the details of the application from the Google Play Store and we also use querying services like DNSBL[2] to check if a server is blacklisted. These network calls combined with drawing the force-directed graph, could increase the wait time for users to see the visualization. To improve this, we do some improvements to the code as stated below.

Global Cache of Application Details: We use BeautifulSoup[18] package for scraping web pages. To get the details of various application, we scrape the Google Play Store web page. This action is expensive in time and takes approximately *4.41 seconds* to get the details of one application. To reduce the amount of time spent on retrieving data, we create a global cache where data related to an application are stored on the server if it was accessed at least once by any user. This brings down the time taken to fetch the details of an application to *0.1 seconds*.



(a) Time-taken in milliseconds to fetch the network traffic data from the database



(b) Time-taken in milliseconds to render the number of edges

Figure 4.5: Graphs showing the time taken to load the visualizations on the server for various simulated conditions

4.5 Example Visualizations

Figure 4.4 is the screen capture of a visualization on the website. This visualization shows the interaction between applications run on the mobile device of a particular user and the remote servers it contacted over the network. As can be seen from the screen shot, this visualization provides the descriptive statistics on how much data was transferred over the network during the requested period of time. Also, this gives the users an insight into how many of the contacted servers were Adservers or trackers. The Adservers or trackers are indicated by a blue circle. The blacklisted servers are indicated by red circles and those servers that are contacted by applications of multiple developers are represented by green ones. This gives users an idea of how much of their data was sent to trackers, Adservers and blacklisted servers.

4.6 Performance Evaluation

Setup: To load the visualizations on the web, the two expensive parts of loading the page are *fetching the data* and *rendering the data*. We evaluate the time taken to load different

amounts of data and we also evaluate the time taken to draw the visualizations.

Results: Figure 4.5(a) plots the time taken to draw various amounts of data. As expected, the amount of data increases the time taken increases, which could increase the loading time of the web page rendering the visualizations. From Figure 4.5(b), we see that when the amount of edges to draw increases, for example to approximately 35000, the time taken increases to about 34 seconds.

Chapter 5

Conclusion

In this thesis, we presented a system to visualize network traffic data - for the first time, in real-time - and to make the network activity on the phone more transparent. We provided a mobile application that builds on and runs along with `AntMonitor` and provides network traffic visualizations on the mobile device in real-time and with negligible delay. We use the data collected from a pilot study to showcase visualization on a web application. We provide insights into the remote servers accessed by mobile application and also indicate remote servers that are accessed by multiple different applications. Table 5.1 summarizes the functionality provided in our visualization on the mobile devices and on the web.

In future work, we would like to be able to identify applications as running in background or foreground and also to visualize PII leaks. We would like to use the visualization to learn user behaviour and for detecting anomalies. Finally, we would like to utilize the power of crowd-sourcing and provide insights and visualizations across multiple users.

Table 5.1: Summary of Visualization Capabilities

Functionality	**Mobile Online**	**Mobile Offline**	**Web/Server Offline**
Real-time Visualization	Yes	No	No
Ad-server identification	No	No	Yes
Blacklist membership	No	No	Yes
Suspicious servers contacted by multiple applications	No	No	Yes
Application icons	Yes	Yes	Yes
Force-directed graph	Yes	Yes	Yes
Historical data	No	No	Yes
Statistics on total data transferred	No	Yes	Yes
Statistics on the amount of data transferred to Labeled nodes	No	No	Yes
Removal of edges on closing connection	Yes	No	No

Bibliography

[1] D3js. https://d3js.org/.

[2] DNSBL. http://www.dnsbl.info/.

[3] List of ad server hostnames. http://pgl.yoyo.org/as/.

[4] PCAPNG File Format. http://goo.gl/y89d9U.

[5] Spamhaus. https://www.spamhaus.org/.

[6] Flashlight. https://play.google.com/store/apps/details?id=com.peacock.flashlight&rdid=com.peacock.flashlight, 2016.

[7] Ghostery. https://play.google.com/store/apps/details?id=com.ghostery.android.ghostery, 2016.

[8] Lightbeam. https://addons.mozilla.org/en-US/firefox/addon/lightbeam, 2016.

[9] Mobile and tablet internet usage surpasses desktop for first time. http://www.zdnet.com/article/mobile-and-tablet-internet-usage-exceeds-desktop-for-first-time-statcounter/, 2016.

[10] Network connections. https://play.google.com/store/apps/details?id=com.antispycell.connmonitor, 2016.

[11] Network log. https://play.google.com/store/apps/details?id=com.googlecode.networklog, 2016.

[12] E. Finickel, A. Lahmadi, F. Beck, and O. Festor. Empirical analysis of android logs using self-organizing maps. In *2014 IEEE International Conference on Communications (ICC)*, pages 1802–1807, June 2014.

[13] M. Kaufmann and D. Wagner. *Drawing graphs: methods and models*, volume 2025. Springer, 2003.

[14] S. G. Kobourov. *Force-directed drawing algorithms. In Handbook of Graph Drawing and Visualization (Discrete Mathematics and Its Applications.* 2013.

[15] A. Lahmadi, F. Beck, E. Finickel, and O. Festor. A platform for the analysis and visualization of network flow data of android environments. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 1129–1130, May 2015.

[16] I. News. Mobile subscriptions near the 7 billion mark. `https://itunews.itu.int/en/3741-Mobile-subscriptions-near-the-78209billion-markbrDoes-almost-everyone-have-a-p` `note.aspx`.

[17] J. Ortiz-Ubarri, H. Ortiz-Zuazaga, A. Maldonado, E. Santos, and J. Grulln. Toa: A web based network flow data monitoring system at scale. In *2015 IEEE International Congress on Big Data*, pages 438–443, June 2015.

[18] L. Richardson. Beautiful soup documentation, 2015.

[19] A. Shuba, A. Le, E. Alimpertis, M. Gjoka, and A. Markopoulou. Antmonitor: System and applications. *arXiv preprint arXiv:1611.04268*, 2016.