ECE 356 Project Report: Internet Traffic

Arjun Bawa – a3bawa

Ricky Li – rjlli

Shane Nesarajah – Sanesara

Repository at https://github.com/arj1211/ece356project/

Introduction

The goal of this project was to design a database architecture to efficiently model relationships between several aspects of internet traffic data. Designing this data source creates a robust framework to query different qualities for real-time network traffic. There were 3 major components of the project. The first was an entity-relationship design. This design modelled the various aspects of internet traffic as entities that interact with each other through relations. This entity-relationship model was then transferred to a relational design and implemented in a MySQL database. Finally, a client application was developed to efficiently add, update, delete, and query information in a user-friendly interface.

Entity Relationship Design

(Bold/Italicize - Entity, Italicize – Relation)

Entities and relationships were added to the model largely based on the structure of the dataset given. They were provided as a csv file where each line represented one flow of data. Given this, it made sense to create an entity representing a network ***Host*** machine, an entity representing a network ***Connection*** between hosts, and a relationship called *Involves* that connects the two. This was because each data flow happens over a connection, and each connection involves a source host and a destination host.

We also realized that a data flow was essentially a compilation of statistical data that applied to a given connection. We created the ***FlowStats*** entity, tracking descriptive statistics of the packets involved in each *flow*

In addition, entities were created to *Log* ***Forward Statistics*** and ***Backward Statistics*** of packets in each flow. Entities describing the ***DirectionalFlagCount*** and ***TotalFlagCount*** presence of flags in packet headers were also created.

Involves Relation Cardinalities

Relationship	Cardinality	Explanation
Host (Source/Destination) to Connection	1:1	A connection is a pairing of source to destination host.
Connection to Host (Source/Destination)	1:1	A connection is a pairing of source to destination host.

Flow Relation Cardinalities

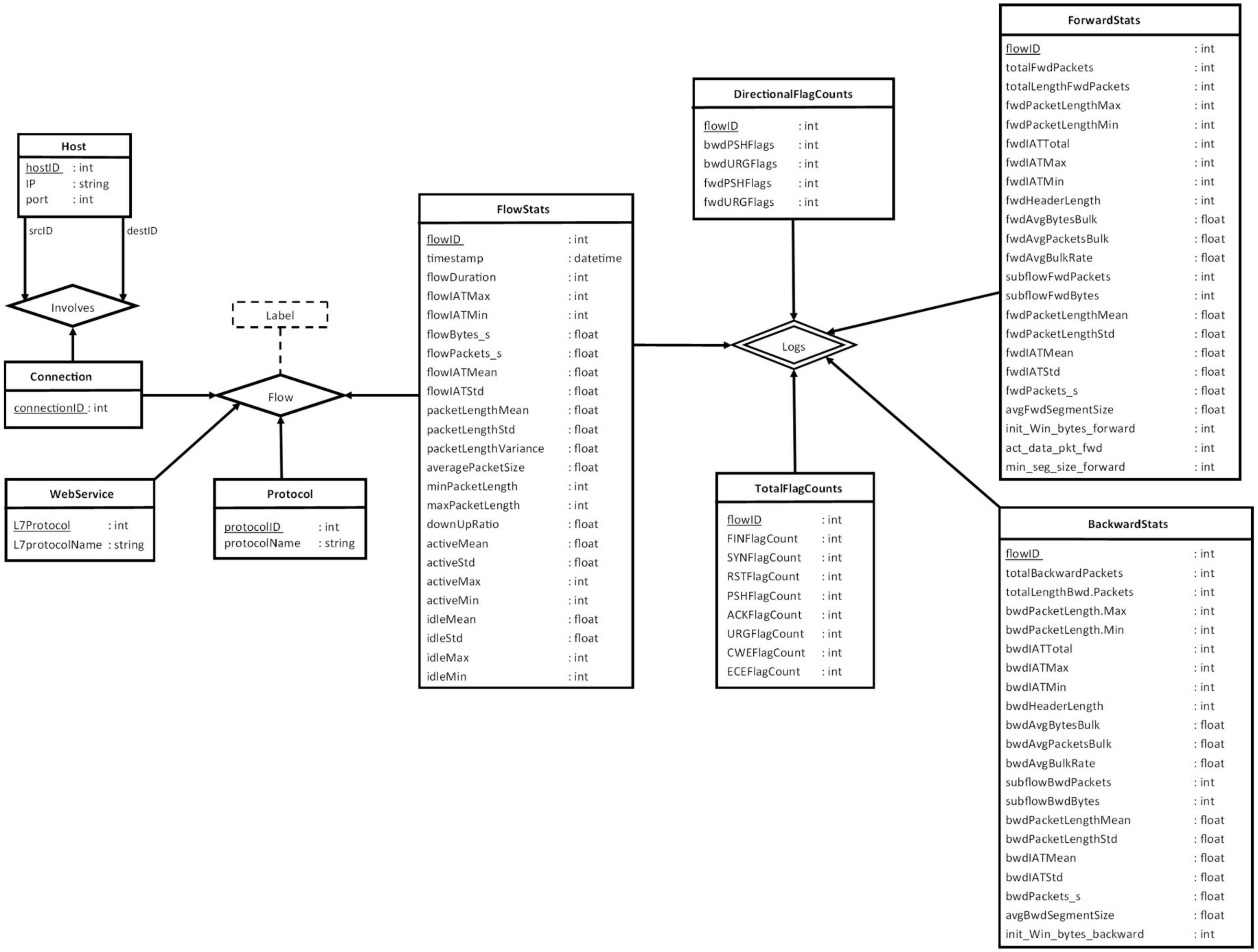
Relationship	Cardinality	Explanation
Connection to FlagInfo	0:*	A connection can have no flows or can have many flows, and each flow has its own flag info.

Connection to FlowStats	0:*	A connection can have no flows or can have many flows.
Connection to protocol	1:*	For two hosts to communicate we need at least one protocol in use. A single connection can have multiple protocols like UDP and TCP.
Connection to Webservice	0:*	A unique connection could access different webservices for example Facebook and Google.
FlagInfo to Connection	1:1	A flag info only has one connection.
FlagInfo to FlowStats	1:1	Each flow has its own flag info.
FlagInfo to Protocol	1:1	Each flow has a single protocol, and the flag info corresponds to a single flow.
FlagInfo to Webservice	1:1	Each flow has a single webservice and the flag info corresponds to a single flow.
FlowStats to connection	1:1	Each flow statistics has a connection.
FlowStats to FlagInfo	1:1	Each flow has its own flag info.
FlowStats to Protocol	1:1	Each flow has its own protocol, so the flow stats describe a single protocol.
FlowStats to Webservice	1:1	Each flow has its own webservice, so the flow stats describe a single webservice.
Protocol to Connection	0:*	Not all protocols will be used by a connection, but it’s possible for a protocol to be used many times.
Protocol to FlagInfo	0:*	Many flag info can use the same few protocols but it’s possible for a protocol to not be used at all.
Protocol to FlowStats	0:*	Many flow stats can use the same few protocols but it’s possible for a protocol to not be used at all.
Protocol to Webservice	0:*	If each webservice has its own protocol then a protocol may not show up in the relation more or less than the number of webservices that use it.
Webservice to Connection	0:*	Webservice to a connection for example google is a webservice that can be accessed by multiple connection and a webservice could possibly have no connections.
Webservice to FlagInfo	0:*	Many flag info can use the same few webservices but it’s possible for a webservice to not be used at all.
Webservice to FlowStats	0:*	Many flow stats can use the same few webservices but it’s possible for a webservice to not be used at all.

Webservice to Protocol	1:N	A specific webservice can only use different protocols.

Logs Relation Cardinalities

All entities in Log are 1:1 for the cardinality because every category of statistics only has one flow.


Host
hostID : int
IP : string
port : int
srcID
destID
Involves
Connection
connectionID : int
Label
Flow
WebService
L7Protocol : int
L7protocolName : string
Protocol
protocolID : int
protocolName : string
FlowStats
flowID : int
timestamp : datetime
flowDuration : int
flowIATMax : int
flowIATMin : int
flowBytes_s : float
flowPackets_s : float
flowIATMean : float
flowIATStd : float
packetLengthMean : float
packetLengthStd : float
packetLengthVariance : float
averagePacketSize : float
minPacketLength : int
maxPacketLength : int
downUpRatio : float
activeMean : float
activeStd : float
activeMax : int
activeMin : int
idleMean : float
idleStd : float
idleMax : int
idleMin : int
DirectionalFlagCounts
flowID : int
bwdPSHFlags : int
bwdURGFlags : int
fwdPSHFlags : int
fwdURGFlags : int
Logs
TotalFlagCounts
flowID : int
FINFlagCount : int
SYNFlagCount : int
RSTFlagCount : int
PSHFlagCount : int
ACKFlagCount : int
URGFlagCount : int
CWEFlagCount : int
ECEFlagCount : int
ForwardStats
flowID : int
totalFwdPackets : int
totalLengthFwdPackets : int
fwdPacketLengthMax : int
fwdPacketLengthMin : int
fwdIATTotal : int
fwdIATMax : int
fwdIATMin : int
fwdHeaderLength : int
fwdAvgBytesBulk : float
fwdAvgPacketsBulk : float
fwdAvgBulkRate : float
subflowFwdPackets : int
subflowFwdBytes : int
fwdPacketLengthMean : float
fwdPacketLengthStd : float
fwdIATMean : float
fwdIATStd : float
fwdPackets_s : float
avgFwdSegmentSize : float
init_Win_bytes_forward : int
act_data_pkt_fwd : int
min_seg_size_forward : int
BackwardStats
flowID : int
totalBackwardPackets : int
totalLengthBwd.Packets : int
bwdPacketLength.Max : int
bwdPacketLength.Min : int
bwdIATTotal : int
bwdIATMax : int
bwdIATMin : int
bwdHeaderLength : int
bwdAvgBytesBulk : float
bwdAvgPacketsBulk : float
bwdAvgBulkRate : float
subflowBwdPackets : int
subflowBwdBytes : int
bwdPacketLengthMean : float
bwdPacketLengthStd : float
bwdIATMean : float
bwdIATStd : float
bwdPackets_s : float
avgBwdSegmentSize : float
init_Win_bytes_backward : int

Relational Schema

Most of our design choices when transferring our ER diagram to a relational schema were fairly obvious. We took all of the entities and turned each one into a table in the database with the exact same primary keys. The only entity which was not turned into a table was the ***Connection*** entity. For this entity, since it is part of the binary relation ***Involves*** and has a 1:1 cardinality, it was subsumed into the Involves table in our database. ***Involves*** and ***Flow*** were the two relationship sets that we had in our ER design. Each one was turned into a table whose columns correspond to the primary keys of the entities that comprised them, with Flow getting a Label column to account for its discriminator.

The one weak entity set in our ER design was ***Logs***. We decided not to turn this into a table because each of the entities that comprise this set have 1:1 cardinalities with each other. We decided that keeping these relations separate in our database but giving each of them a primary key that would allow them to easily be associated with each other (flowID) was a more user-friendly design choice than combining them into one huge table. When querying statistical data for a given flow, this allows the user to either access data from a specific table, or access data from *all* tables, easily and efficiently.

We decided not to add any indexes because the ones automatically created by SQL when specifying primary key and unique constraints were sufficient for the expected queries that would be carried out over the schema. For example, we specified that the Host table would have a unique constraint over (IPAddress, portNumber). Our client application does not allow for querying over portNumber, but it does for querying over IPAddress. It also requires us to run queries over the Involves table. This index increases the efficiency of queries run by the user while also increasing the efficiency of matching a sourceID or destinationID from the Involves table to a hostID.

Tests were run to validate the design. This testing included inserting, deleting, and updating data into the database and making sure that all constraints were followed. For example, a user should not be able to create statistics for a flow that does not exist, and in order for a flow to exist, it must have run over a connection using a certain protocol and web service. So, we tested to make sure that forwardStatistics could not be added to unless the user had a corresponding flowID, which referenced the flowData table. We also ensured that there was no information lost in the transition from the original dataset to our schema. An example of these tests was when we checked how many flows had connected to each web service, and then added the results up. The expected sum was the number of lines we read from the data set, but the observed result was hundreds of times higher. Eventually we realized this was because of some mistakes in the sql query we were using to run this test. We fixed the issue with the join syntax, and the new observed result was equal to the expected one.

Host(<u>hostID</u>, IPAddress, portNumber)

Involves(<u>connectionID</u>, sourceID, destinationID)

- sourceID and destinationID both reference Host.hostID

WebService(L7Protocol, L7protocolName)

Protocol(protocolID, protocolName)

FlowStats(flowID, timestamp, flowDuration, flowIATMax flowIATMin, flowBytesS, flowPacketsS, flowIATMean, flowIATStd, packetLengthMean, packetLengthStd, packetLengthVariance, averagePacketSize, minPacketLength, maxPacketLength, downUpRatio, activeMean, activeStd, activeMax, activeMin, idleMean, idleStd, idleMax, idleMin)

Flow(flowID, connectionID, protocolID, L7protocol, label)

- flowID references FlowStats.flowID
- connectionID references Involves.connectionID
- protocolID references Protocol.protocolID
- L7protocol references WebService.L7protocol

DirectionalFlagCounts(flowID, bwdPSHFlags, bwdURGFlags, fwdPSHFlags, fwdURGFlags)

- flowID references FlowStats.flowID

TotalFlagCounts(flowID, FINFlagCount, SYNFlagCount, RSTFlagCount, PSHFlagCount, ACKFlagCount, URGFlagCount, CWEFlagCount, ECEFlagCount)

- flowID references FlowStats.flowID

ForwardStats(flowID, totalFwdPackets, totalLengthFwdPackets, fwdPacketLengthMax, fwdPacketLengthMin, fwdIATTotal, fwdIATMax, fwdIATMin, fwdHeaderLength, fwdAvgBytesBulk, fwdAvgPacketsBulk, fwdAvgBulkRate, subflowFwdPackets, subflowFwdBytes, fwdPacketLengthMean, fwdPacketLengthStd, fwdIATMean, fwdIATStd, fwdPackets_s, avgFwdSegmentSize, init_Win_bytes_forward, act_data_pkt_fwd, min_seg_size_forward)

- flowID references FlowStats.flowID

BackwardStats(flowID, totalBackwardPackets, totalLengthBwdPackets, bwdPacketLengthMax, bwdPacketLengthMin, bwdIATTotal, bwdIATMax, bwdIATMin, bwdHeaderLength, bwdAvgBytesBulk, bwdAvgPacketsBulk, bwdAvgBulkRate, subflowBwdPackets, subflowBwdBytes, bwdPacketLengthMean, bwdPacketLengthStd, bwdIATMean, bwdIATStd, bwdPackets_s, avgBwdSegmentSize, init_Win_bytes_backward)

- flowID references FlowStats.flowID

Client Application

(a) Ideal client requirements

The ideal client application would allow users to signify what level of access they require, along with security measures for the higher levels of access (e.g. password). The different access levels would dictate what actions the user can take in the database and to what degree. For example, a general user may be able to add data to only a few tables, while an administrator may be able to add data to any table. The application should also allow for querying the data based on attributes that a customer in the domain would often require. For internet traffic data, this could include source/destination IP, webservice, or protocol.

1. Querying the data in a way that a customer in the domain would do

- Source ip, destination ip
- Webservice
- Timestamp (month, year, day)
- Flow bytes/s
- Active/idle stats

2. Modifying the data in a way that a customer in the domain would do

- Add a flow (only to flow stats, or to flowstats and one or more of the other log tables)
    - Must add to flow table
- Add a protocol
- Add a webservice
- Add a host
- Add a connection

(b) Actual client proposed

- User can select from one of two levels of access: Regular or Administrator
    - Admin requires password
- Administrator
    - Can query information from any tables he selects
    - Can query based on the columns:
        - Source and destination ip and port
        - Webservice, protocol
        - Any column in the entire database
    - Can add to or modify any table (while following constraints)
- Regular user
    - Can query information from Involves, Webservice, Protocol, FlowStats, and Flow
    - Can query based on the columns:
        - Source and destination ip
        - Webservice, protocol, timestamp
    - Can add webservice, protocol, or host
    - Cannot modify data

(c) Actual client implemented and justification for design

- User can select from one of two levels of access: Regular or Administrator
    - Admin requires password
- Administrator
    - Can choose from a predetermined list of queries
        - Queries are separated based on which attributes they filter by
    - Can query based on the columns:
        - Source and destination host
        - Webservice
        - Protocol
    - Can add to or delete from any table (while following constraints)
- Regular user
    - Can choose from a predetermined list of queries
        - Queries are separated based on which attributes they filter by
    - Can query based on the columns:
        - Source and destination ip
        - Webservice
        - protocol
    - Can add webservice, protocol, or host
    - Cannot modify data

The actual client we implemented was different but still very similar to the proposed client. The biggest difference is that the actual client implemented makes the user choose from a predetermined list of possible queries instead of selecting themselves which tables and which attributes they would like to query from. It was decided that this was the most user-friendly format for the interface, mainly due to the large number of tables and attributes in the database. We realized that most queries run by a user would not be looking for individual statistics regarding each flow, but instead for the relationships between different entities, such as a list of hosts who have connected to each webservice, or the number of flows that used a certain protocol. And if they did want flow statistics, choosing between large groupings of statistics rather than individual was more useful. The list of predetermined queries is presented to the user separated by the attributes they filter by.