

Volunteer-Based Distributed Traffic Data Collection System

Kartheepan Balachandran, Jacob Honoré Broberg, Kasper Revsbech, Jens Myrup Pedersen
Aalborg University, Department of Electronic Systems, DK-9220, Aalborg East, Denmark
{bkarthe, jbroberg, kar, jens}@es.aau.dk

Abstract—An architecture for a traffic data collection system is proposed, which can collect data without having access to a backbone network. Contrary to other monitoring systems it relies on volunteers to install a program on their own computers, which will capture incoming and outgoing packets, group them into flows and send the flow data to a central server. Data can be used for studying and characterising internet traffic and for testing traffic models by regenerating real traffic. The architecture is designed to have efficient and light usage of resources on both client and server sides. Worst case performance estimates indicate that this is obtained. Tests conducted by volunteers using an implemented prototype confirm the feasibility of the system.

Index Terms—Internet, traffic generating, traffic collecting, traffic monitoring, packet inspection, volunteer-based system

I. INTRODUCTION

Studying real-life internet traffic is becoming an increasingly complex task, due to proprietary application protocols and P2P traffic (some of which disguises itself as port 80 traffic) which are harder to characterise than the traditional protocols based on single TCP-sessions and static port numbers registered by IANA. The importance of understanding the behaviour and pattern of concurrent IP traffic is increasing in order to provide good QoS and performance in network planning, to understand and circumvent security breaches and to create realistic traffic models and generators for system testing. To do any of this real internet traffic must be collected and studied, which is often done by capturing packets in a ISP/corporate/campus backbone network and analyse it online or offline as a trace. However, getting access to any of these facilities can be difficult due to legal, privacy and access rights concerns, which is the problem addressed here. This is why we propose a volunteer based traffic data collection system.

The purpose of this system is to let ordinary internet users volunteer to allow the internet traffic on their private PCs to be monitored. The volunteer will download and install a data collecting program. This program will run as a background process, collecting the traffic data and send it to a server, which stores the data for offline analysis carried out by various researchers. We believe that volunteers for such a system can be recruited from the ever-growing internet community, based on the motivation of contributing to scientific research, as was the case with e.g. the SETI@home project [1], which has had several million volunteer users. The incentive to

contribute could be to improve the internet and supply higher QoS to end-users. Volunteering for a project like this can be compared with accepting to participate in a poll. According to [2] volunteer computing has a huge potential, not only measured in potential CPU-cycles.

In traditional passive single-point monitoring systems data is often stored as flows based on the 5-tuple definition (*source IP, destination IP, source port, destination port, protocol no.*) as in [3] and [4]. Systems making data available for multiple users have also been proposed, such as [5] and [6]. The latter, the European distributed monitoring system LOBSTER, gives access to authenticated user groups using DiMAPI [7]. Another distributed monitoring system is DOME [8], which avoids the bulky trace database by letting each measurement node do the processing itself based on user queries. Contrary to the proposed system all of these systems rely on access to backbone-like networks to capture traffic and most of the systems are also focused on real-time/online analysis.

The purpose of the proposed system is to collect traffic data, which researchers can analyse using appropriate analysis schemes. Nevertheless the current traffic analysis schemes have been studied for the architectural design. Previously, all traffic analysis was based on port numbers such as in CoralReef [9]. In [10] a method was proposed to analyse application level traffic by grouping flows according to relationships based on port numbers. This method was later extended with signature mapping [11]. In [12] a comparison is made between three different types of application layer traffic classification methods based on ports, host behaviour and flow features. Deep packet inspection is also used for analysis [13].

The scope of the paper is to outline an architecture for a data collection system suitable for volunteer usage with focus on the measurement node application. In Section II the terminology is outlined. In Section III system requirements concerning privacy and performance are discussed. In Section IV the system architecture is described, which is based on a modular design allowing use of state-of-the-art components for packet capturing, payload analysis, data anonymisation and compression. In Section V worst case estimates and prototype test results are presented to argue that the system design is feasible. Section VI briefly concludes the paper.

II. TERMINOLOGY

Here the terminology describing the system actors is listed: **Volunteer** - the ordinary internet user allowing his computer to be monitored. **Measurement node** - a computer of a volunteer. **User** - a researcher using the collected data for analysis or testing purposes. **Server** - the central repository of collected data accessible for the users. **Flow** - an abstraction to group packets on the measurement node.

Packets captured on a measurement node are grouped into flows according to a 5-tuple definition (*local IP, remote IP, local port, remote port, protocol no.*). A flow is defined as bidirectional traffic between two host ports, since all bidirectional traffic will be captured with certainty when data is collected at the end user, contrary to backbone data collection. Hence, the source/destination notation cannot be used, because the source and destination IP and port pair of a packet can be swapped and still belong to the same flow. Instead local/remote notation is used, where local is the IP and port on the measurement node. A flow is created when a packet not belonging to any open flow is captured. A flow is closed if it has not been updated with any packets for a given time interval (i.e. a timeout).

III. SYSTEM REQUIREMENTS

A. Volunteer Privacy and Anonymity

Respecting the privacy and anonymity of the volunteer is important, since very few would volunteer if their private data was logged and sent to a remote server. Hence, it would limit the distribution of the system. Private data is that which can be used to trace a specific host, such as IP addresses. Furthermore, application payload is considered to be private data since it can contain unencrypted usernames and passwords and private messages in general. However, many traffic classification techniques rely on deep packet inspection. Such analysis has to be carried out on the measurement node as in [8] and only the analysis results can be sent to the server. IP addresses can be anonymised using one of the standard methods such as prefix preserving anonymisation [14].

B. Measurement Node Resource Usage

The resource usage on the measurement node must be kept low and controlled, otherwise people will not volunteer. The resources in question are: CPU power, memory, disk storage and bandwidth. The resource usage should be minimised by making the software light. Furthermore, heavy processing or bandwidth consuming tasks should be scheduled to avoid annoying the volunteer, like SETI@home which runs when the computer is in screensaver mode [1]. However, if doing so, memory and storage usage should be considered. To reduce bandwidth usage the data sent to the server can also be compressed e.g. using the method described in [15].

C. Server Resource Usage and Scalability

The server should have a high availability to ensure a robust system. It must be able to handle multiple connections from many clients (i.e. measurement nodes) when they request to transmit data. The number of simultaneous connecting measurements nodes is proportional to the available bandwidth of the server. The server must have enough resources in terms of disk storage, CPU power and memory to its disposal in order to keep the robustness of the whole system. The disk storage needed depends on the flow format and amount of the data sent from the measurement nodes. If a single server does not have enough resources, it must be possible to add more servers to work in a cluster.

D. System Extendibility and Feasible Constant Improvement

The proposed system is the first release of a system which can be improved and extended for different purposes. In the architectural design it must therefore be ensured that each component is modular and has well defined interfaces to allow for improvements. The measurement node software must be designed so that volunteers only need to allow for regular updates, like virus scanners update regularly.

IV. SYSTEM ARCHITECTURE

A. Architecture of Measurement Node

The purpose of the system is to collect packets while the volunteer is using the internet connection. The architecture of the measurement node is divided into modules shown in Figure 1, which is described in the following. The numbered arrows describe the interfaces between modules. The modular design allows for updates as described in Section III-D.

1) *Packet Capturer*: This module captures all IP packets sent and received from the network interface connected to the internet. The captured packets are timestamped and put in a queue for the Flow Generator represented by interface 1.

2) *Socket Monitor*: This module maintains a list of opened and closed sockets. For each socket the name of the application is recorded together with port no. and timestamps for when the application opened and closed the socket. This list is used to classify flows combined with payload analysis. The information is given through interface 2 and could also be used to determine when a flow is created and closed.

3) *Flow Processing*: The *Flow Generator* in this frame groups packets into flows. Packets are retrieved through interface 2 and the 5-tuple information for each packet is extracted. If an open flow matching this information already exists, the packet is added to the flow. If not, a new flow is created for the packet. The packet payload is passed to the *Payload Analyser* through interface 3 along with information about which flow the packet belongs to. When a flow has been closed it is passed to the *Flow Anonymiser* through interface 4, where IP addresses are replaced. The anonymised

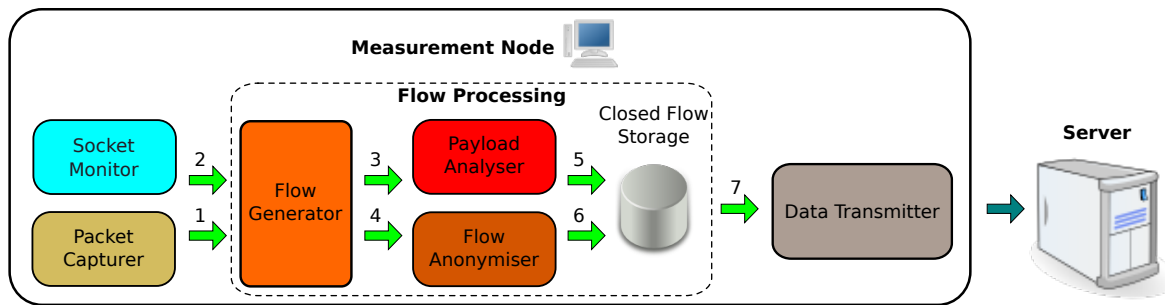


Figure 1. The architecture of the measurement node software, i.e. the system running on the volunteer computer collecting traffic data.

flow is then passed to the *Closed Flow Storage* through interface 6. When the Payload Analyser has processed the payload data the analysis result is added to the corresponding flow in the Closed Flow Storage through interface 5.

4) *Data Transmission*: This module controls when to transmit closed flow data retrieved through interface 7 to the server. Data is compressed before transmission. Transmission can be scheduled to run when bandwidth is available.

B. Architecture of Server

All measurement nodes transfer their flow data to the server. The architecture of the server is shown in Figure 2.

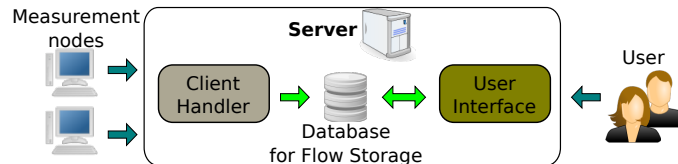


Figure 2. The architecture of the server. It has two interfaces, one towards the measurement nodes and one for users.

The *Client Handler* services all incoming connections from known clients and receives the transmitted data and stores the data in the database. When users want to analyse the data they send queries to the database through the *User Interface*.

C. Flow Record Format

For each flow a record is saved similar to that of other traffic monitoring systems [3][4]. As shown in Figure 3 the general flow information is saved in the flow header and the specific information for each packet is saved in packet entries.

The 5-tuple information is saved in the flow header. For non TCP/UDP traffic the port numbers are omitted, illustrated with the marked fields in Figure 3. The flow header also contains the timestamps of the flow, the no. of packets in the flow, and fields for both the socket application name and the payload analysis result to classify the flow. These fields can vary in size and possibly be omitted by setting the length;

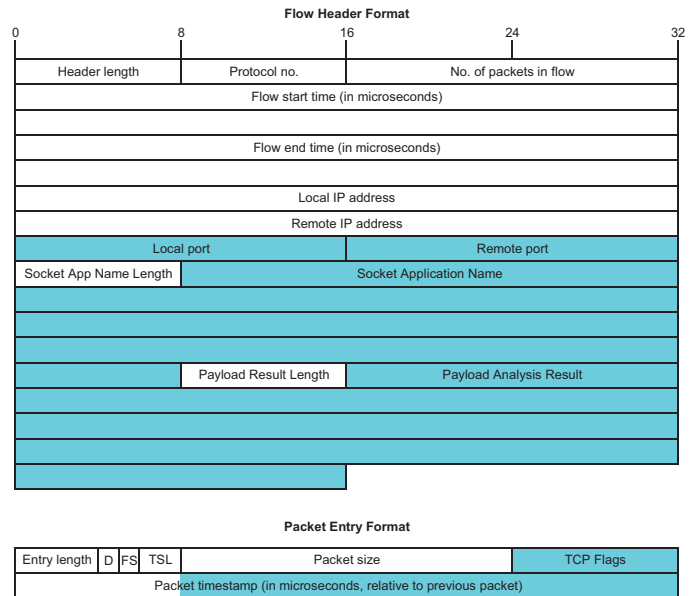


Figure 3. The format of the flow header and the packet entry, the two parts of the flow record. The marked areas can be omitted.

however, it is believed that 16 bytes for each will be sufficient.

The size and relative timestamp of each packet is saved in the packet entry, so that packets can be accurately regenerated and used for testing traffic models. The timestamp is relative to the timestamp of the previous packet in the flow and both 8, 16, 24 and 32 bits can be used depending on the length needed, which is set using the two bit TimeStamp Length (TSL) field. Furthermore, if the flow is TCP, the TCP flags are stored and the Flags Set (FS) bit is set. The Direction (D) of the packet is also set, i.e. local→remote or remote→local.

D. Prototype Implementation

A prototype of the measurement node application has been implemented in Java to use for performance tests conducted by volunteers. The Packet Capturer has been implemented using the Jpcap library [16], a Java wrapper to libpcap/Winpcap. The Socket Monitor, Payload Analyser and Flow Anonymiser have only been implemented as stubs. In

the tests a number of statistics have been saved and processed using Matlab. These statistics will be discussed in Section V.

V. ESTIMATES AND RESULTS

A. Packet Capture Estimates and Results

In the following it is illustrated how much traffic data a measurement node will capture and process, based on worst-case estimates and volunteer tests using the implemented prototype. It is assumed that a regular volunteer has a 4 Mbit/s download and 512 Kbit/s upload internet connection and is connected with regular Ethernet. The estimated amount of captured traffic depends on the link net speed on the IP layer (v_{net}), which can be calculated as follows:

$$v_{\text{net}} = \frac{s_{\text{payload}}}{s_{\text{payload}} + s_{\text{overhead}}} \cdot (v_{\text{up}} + v_{\text{down}}) \quad [\text{bit/s}] \quad (1)$$

where $v_{\text{up}} + v_{\text{down}}$ is the sum of both upload and download speed (4512 Kbit/s), s_{payload} is the size of the Ethernet payload, s_{overhead} is the Ethernet overhead (38 bytes).

In Table 1 the worst-case amount of captured data is shown assuming full net speed usage. The payload size (s_{payload}) of the max frame size estimate is the Ethernet MTU (1500 bytes) and 46 bytes for the min frame size estimate. Furthermore, eighth tests of three hours each have been carried out by different volunteers of which the results of four are shown in Table 1. These four tests have been chosen because they represent the different usage patterns in all the tests. The tests with their given PC specifications will be referred to as Test 1-4 in the rest of the paper.

Table 1. Estimates and Test Results Showing the Amount of Captured Data.

	Amount of captured data per second	Amount of captured data per hour	Average packet size
Estimate 1: Full bandwidth usage* max frame size	537 KB	1889 MB	1500 bytes
Estimate 2: Full bandwidth usage* min frame size	302 KB	1060 MB	46 bytes
Test 1: 1.4 GHz, 958 MB RAM	2.3 KB	8.2 MB	304 bytes
Test 2: 2 GHz, 893 MB RAM	11.4 KB	40 MB	740 bytes
Test 3: 1.7 GHz, 2 GB RAM	28.7 KB	101 MB	838 bytes
Test 4: 1.8 GHz, 3 GB RAM	161 KB	565 MB	612 bytes

*IP layer net speed: 4.40 Mbit/s for max frame size, 2.47 Mbit/s for min frame size

The estimates in Table 1 show that a large amount of data will be captured in worst-case by the Packet Capturer both for max size frames and min size frames. If the flow grouping of raw packets was postponed to a more suitable time (in terms of available CPU resources), this large amount of raw data would use an excessive amount of memory. Hence, the Flow Generator will instantly process captured packets. The test results show a more moderate amount of captured data

varying according to usage pattern. These results confirm that the worst case estimates should be viewed as upper bounds.

B. Flow Estimates and Results

In the following a number of worst-case estimates and test results are presented illustrating how much flow record data will be generated and the time it takes to transfer this data to the server, assuming uncompressed transfer.

Depending on the average amount of packets in a flow ($\text{avg}\{p/f\}$), the accumulated size of flow record data ($\sum s_{\text{fr}}$) can be estimated as follows:

$$\sum s_{\text{fr}} = \sum p \cdot (s_{\text{fh}} + s_{\text{pe}} \cdot \text{avg}\{p/f\}) \quad [\text{bytes}] \quad (2)$$

where $\sum p$ is the total no. of packets, s_{fh} is the flow header size and s_{pe} is the packet entry size (which are max 66 bytes and 8 bytes, see Figure 3). The time it takes to transfer data to the server (t_{trans}) can be calculated as follows:

$$t_{\text{trans}} = \frac{\sum s_{\text{fr}}}{v_{\text{up,net}}} \quad [\text{sec}] \quad (3)$$

where $v_{\text{up,net}}$ is the upload net speed. Only half of the available bandwidth is used to avoid using all the measurement node bandwidth resources. This is 256 Kbit/s, which gives a net speed of 243 Kbit/s on the application layer level, when Ethernet, IP and TCP headers are subtracted and max size frames are assumed.

Table 2. Estimates and Test Results Showing Flow Record Statistics.

	Average packets per flow	No. of flows per hour	Flow record data per hour**	Transmission time of one hour of flow record data***
Estimate 1	1	1,320,156	93.2 MB / 4.93 %	54 min
Estimate 2	1	24,171,429	1706 MB / 90.3 %	981 min (16.4 h)
Test 1	8	3,357	374 KB / 4.43 %	13 sec
Test 2	26	2,173	445 KB / 1.11 %	16 sec
Test 3	92	1,371	787 KB / 0.76 %	27 sec
Test 4	77	12,446	5.9 MB / 1.04 %	203 sec
Web browsing*	46	3,905	1.2 MB / 1.00 %	41 sec
YouTube*	374	1,423	2.86 MB / 0.69 %	99 sec
FTP*	42	6,293	1.84 MB / 1.54 %	64 sec
Email*	81	335	174 KB / 1.17 %	6 sec
Skype*	158	928	729 KB / 3.81 %	27 sec
BitTorrent 1*	33	8,609	2.2 MB / 1.53 %	75 sec
BitTorrent 2*	642	1,478	5.4 MB / 0.84 %	186 sec

*Based on 30 min tests using only that application

**Total size of record data / ratio between record data size and amount of captured traffic

***Assuming 256 Kbit/s bandwidth usage = 243 Kbit/s app. layer net speed

Table 2 shows a number of estimated and measured flow record statistics. The two estimates use the results from Table 1 to illustrate the absolute worst case situation, i.e. to have an average of 1 packet/flow and full bandwidth usage. The results for the four tests (described in Table 1) have been derived based on the measured no. of packets and flows and size of flow record data. Furthermore, the results of seven 30 min application specific tests conducted using the implemented prototype are shown to illustrate the flow

characteristics when limiting internet usage to one of these common applications.

As the estimates in Table 2 show, both max size frames (Estimate 1) and min size frames (Estimate 2) for 1 packet/flow result in a large amount of data to save and a long transmission time. This is particularly true for Estimate 2, however, assuming full bandwidth usage with almost empty packets is not very likely. Furthermore, the test results show a higher average no. of packets per flow and in turn a smaller amount of flow record data and a shorter transmission time. In Test 1 the average no. of packets per flow is low, but the amount of captured traffic data is also low, whereas more traffic in Test 3 and Test 4 result in a higher average no. of packets per flow. Both Estimate 1 and the test results show that a small size ratio between the captured traffic and the flow record data is achieved. All of this indicates that data can be stored and transferred as flow records without depleting measurement node resources.

C. Payload Analysis Estimates and Results

For each flow a sufficient amount of packet payloads must also be analysed by the Payload Analyser to characterise the application layer traffic along with the information from the Socket Monitor. In the following it is estimated how much payload data will have to be processed in worst-case and the measured amount of data from the volunteer tests is shown.

All the traffic data described in Section V-A does not have to be analysed. It is sufficient to analyse the first 100 bytes of payload data according to [13] and the first 16 bytes according to [12]. In [11] good results were obtained by analysing the first 100 bytes of the first 5 packets in each flow, using a 5-tuple flow definition similar to the definition used here. It is therefore assumed to be sufficient to process 100 bytes of the first 5 packets in a flow, i.e. between 100-500 bytes per flow.

Table 3. Estimates and Test Results for Amount of Payload Data to Process.

	Payload data to process per second*	Payload data to process per hour*	Payload data to process* / captured traffic data
Estimate 1	36 KB	126 MB	6.67 %
Test 1	201 bytes	706 KB	8.37 %
Test 2	67 bytes	235 KB	0.56 %
Test 3	46 bytes	163 KB	0.16 %
Test 4	411 bytes	1.4 MB	0.25 %

*Processing only the first 100 bytes of the first 5 packets in each flow

Table 3 shows the estimated amount of payload data for max size frames (Estimate 1) using the no. of flows from Table 2, along with the measured amount of payload data for the four volunteer tests. The results show that only a small part of the captured traffic has to be analysed. Furthermore, given this small amount of payload data, it is feasible to schedule payload analysis, if CPU resources do not allow instant processing, and thus temporarily store the payload

data. However, it is of course preferable to do payload analysis instantly and thereby avoid memory/disk usage.

D. CPU and Memory Usage

The resource usage of the measurement node application should be kept low. To evaluate this, a resource monitor has been included in the implemented prototype. Every 5 seconds the monitor samples the time spend by the CPU to run any of the threads in the Java Virtual Machine (JVM) executing the application and the memory consumption of the JVM. The CPU usage in percentage is derived by comparing the CPU time with the actual time spend (wall time). The results for the four volunteer tests are shown in Table 4.

Table 4. CPU and Memory Usage for the Volunteer Tests.

	Average CPU usage*	Peak CPU usage*	Average memory usage**	Peak memory usage**
Test 1	0.09 %	1.25 %	16.0 MB	18.0 MB
Test 2	0.31 %	3.12 %	15.8 MB	18.4 MB
Test 3	0.44 %	4.37 %	16.1 MB	18.2 MB
Test 4	0.76 %	5.23 %	30.16 MB	44.4 MB

*Based on the total CPU time spend by the Java Virtual Machine running the test program

**Based on the total memory (both heap and non-heap) by the Java Virtual Machine running the test program

The results show a low average CPU usage and an acceptable peak CPU usage for all tests. The memory usage, both average and peak, is also acceptable when taking the computer specifications shown in Table 1 into account. Nevertheless, since this is a prototype it is assumed that resource usage can be lowered by optimising the implementation. However, the results confirm that a light-weight implementation is feasible.

E. Server Estimates

In the following the estimated and measured flow record data sizes from Table 2 are used to estimate how the server should be scaled in order to store the received data for offline analysis. It is assumed that the server has a 100 Mbit/s connection, which can be used for receiving data from the measurement nodes. Assuming that each measurement node uploads with a speed of 256 Kbit/s (and ignoring control communication), the server can handle 390 simultaneous data transfers from measurement nodes connecting to the server at the same time.

To test the upper limit imposed by the server CPU capacity a test has been conducted, where 10 PCs constantly transfer data with 256 Kbit/s to a server (with specifications: dual core 2.2 GHz, 3 GB RAM, 5200 RPM hard disk), resulting in an average CPU usage of 2.47 % on the server. Based on these results it is estimated that no more than 404 measurement nodes can be simultaneously connected to a single server with the mentioned specifications, however the real limit will probably be lower due to limitations of disk I/O in particular.

This limit can be raised if more servers work in a cluster. However, the total no. of measurement nodes, which can be supported, will be higher, since they do not all transfer data at the same time, but only for a short period (cf. Table 2).

Table 5. Estimates and Test Results for Server Storage Needs.

	Storage need for one hour of data collection*	Storage need for one day (8 hours) of data collection*	Storage need for one week (7 days) of data collection*
Estimate 1	93 MB / 9.1 GB	745 MB / 73 GB	5.1 GB / 510 GB
Estimate 2	1.7 GB / 167 GB	13.3 GB / 1.3 TB	93 GB / 9.1 TB
Test 1	374 KB / 36.5 MB	2.9 MB / 292 MB	20.4 MB / 2 GB
Test 2	455 KB / 44.5 MB	3.5 MB / 356 MB	24.9 MB / 2.4 GB
Test 3	787 KB / 76.9 MB	6.2 MB / 615 MB	43.1 MB / 4.2 GB
Test 4	5.9 MB / 589 MB	47.1 MB / 4.6 GB	330 MB / 32.2 GB

*1 measurement node / 100 measurement nodes

In Table 5 requirements for server storage are illustrated using both estimates and test results from Table 2 for both 1 and 100 measurement nodes. A day is estimated as 8 hours of internet usage. In the worst-case scenarios where the full bandwidth is used on empty packets for all the 100 measurement nodes, the server has to save up to 9.1 TB per week. However, in a more real-life scenario the server only has to store around 32.2 GB per week according to the tests results, which is definitely feasible with modern storage capacities. With these storage needs data can easily be stored for one or more weeks.

VI. CONCLUSION

In this paper we have presented an architecture proposal for a new way of collecting traffic by using ordinary internet users as volunteers. This way internet traffic studies can be conducted without having access to a backbone network. To ensure that volunteers will actually install a monitoring program, their privacy must be protected and a low CPU, memory, bandwidth and disk usage must be guaranteed.

The proposed architecture is divided into modular components, allowing each of the components to be individually improved. Captured traffic is grouped and stored as flows according to a bidirectional 5-tuple definition. Payload is analysed locally on the measurement node (the computer of the volunteer) due to privacy concerns. The name of the application opening the socket is recorded. The collected data is compressed and transferred to the server. Based on worst-case estimates and prototype test results it has been shown how much internet traffic an ordinary volunteer can and will capture and process. The estimates have been shown to be several magnitudes higher than the test results. Both estimates and test results show that it is feasible to run a traffic data collecting system with minimal resource usage and that a server can robustly handle several hundreds of measurement nodes and store collected data.

VII. ACKNOWLEDGMENT

The research work was conducted in the context of a master student project at the Department of Electronic Systems at Aalborg University, and as such supported by the Study Board at Aalborg University, see www.esn.aau.dk.

REFERENCES

- [1] E. Korpela, D. Werthimer, D. Anderson, J. Cobb, and M. Leboisky, "SETI@home - massively distributed computing for SETI," *Computing in Science & Engineering*, vol. 3, no. 1, pp. 78–83, Jan/Feb 2001.
- [2] D. Anderson and G. Fedak, "The computational and storage potential of volunteer computing," in *CCGRID '06: Sixth IEEE International Symposium on Cluster Computing and the Grid*, 2006, vol. 1, May 2006, pp. 73–80.
- [3] G. Iannaccone, C. Diot, I. Graham, and N. McKeown, "Monitoring very high speed links," in *IMW '01: Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*. New York, NY, USA: ACM, 2001, pp. 267–271.
- [4] S.-H. Han, M.-S. Kim, H.-T. Ju, and J. W.-K. Hong, "The architecture of NG-MON: A passive network monitoring system for high-speed IP networks," in *DSOM '02: Proceedings of the 13th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*. London, UK: Springer-Verlag, 2002, pp. 16–27.
- [5] A. Hussain, G. Bartlett, Y. Pryadkin, J. Heidemann, C. Papadopoulos, and J. Bannister, "Experiences with a continuous network tracing infrastructure," in *MineNet '05: Proceedings of the 2005 ACM SIGCOMM workshop on Mining network data*. New York, NY, USA: ACM, 2005, pp. 185–190.
- [6] D. Antoniadis, P. Trimintzios, M. Polychronakis, S. Ubik, A. Papadogiannakis, and V. Smotlacha, "LOBSTER: a European platform for passive network traffic monitoring," in *TridentCom '08: Proceedings of the 4th International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities*, 2008, pp. 1–10.
- [7] P. Trimintzios, M. Polychronakis, A. Papadogiannakis, M. Foukarakis, E. P. Markatos, and A. Øslebø, "DiMAPI: an application programming interface for distributed network monitoring," in *NOMS '06: Proceedings of the 10th IEEE/IFIP Network Operations and Management Symposium*, April 2006.
- [8] T. Wolf, R. Ramaswamy, S. Bunga, and N. Yang, "An architecture for distributed real-time passive network measurement," in *MASCOTS '06: Proceedings of the 14th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*. Los Alamitos, CA, USA: IEEE Computer Society, 2006, pp. 335–344.
- [9] CAIDA: CoralReef, 2009. [Online]. Available: <http://www.caida.org/tools/measurement/coralreef/>
- [10] M.-S. Kim, Y. J. Won, and J. W.-K. Hong, "Application-level traffic monitoring and an analysis on IP networks," *ETRI Journal*, vol. 27, no. 1, pp. 22–42, February 2005.
- [11] Y. Won, B.-C. Park, H.-T. Ju, M.-S. Kim, and J. Hong, "A hybrid approach for accurate application traffic identification," in *2006 4th IEEE/IFIP Workshop on End-to-End Monitoring Techniques and Services*, April 2006, pp. 1–8.
- [12] H. Kim, K. Claffy, M. Fomenkov, D. Barman, M. Faloutsos, and K. Lee, "Internet traffic classification demystified: Myths, caveats, and the best practices," in *CONEXT '08: Proceedings of the 2008 ACM CoNEXT Conference*, 2008, pp. 1–12.
- [13] D. Antoniadis, M. Polychronakis, S. Antonatos, E. P. Markatos, S. Ubik, and A. Øslebø, "Appmon: An application for accurate per application network traffic characterisation," in *Proceedings of IST Broadband Europe 2006 Conference*, December 2006.
- [14] J. Xu, J. Fan, M. H. Ammar, and S. B. Moon, "On the design and performance of prefix-preserving IP traffic trace anonymization," in *Internet Measurement Workshop*, 2001, pp. 263–266.
- [15] P. Politopoulos, E. Markatos, and S. Ioannidis, "Evaluation of compression of remote network monitoring data streams," in *Network Operations and Management Symposium Workshops*, 2008. NOMS Workshops 2008, April 2008, pp. 109–115.
- [16] Jpcap, Java library, 2009. [Online]. Available: <http://netresearch.ics.uci.edu/kfujii/jpcap/doc/>