UiO1 – Web performance in practice – "Why we are waiting?" – Ten years after

Responsible: Matti Siekkinen, University of Oslo, siekkine@ifi.uio.no

Overview

The objective of this assignment is to find out why do we wait after clicking or typing a new address on our browser. In other words, we want to find out where the waiting time is spent (DNS resolving, HTTP transaction overhead, data transfer, server/browser processing...). The idea is to see whether measurements taken ten years after still corroborate the results presented in [1] where the authors used trace from 1998 and 1999, or whether there have been important changes. Indeed, a lot has changed in ten years with respect to processing power, available bandwidth, content offered in the web, content distribution methods (server farms etc.). It is interesting to see how this affects everyday users web surfing experience.

Tasks to do:

- establish necessary measurements infrastructure on a single host
    - tcpdump/wireshark
    - other tools like web request analyzers etc.
- surf the web
    - select different kinds of sites (e.g. news portals, home pages, www.youtube.com...)
    - browse automatically via scripts and wget, for instance, or manually
- analyze measurement results

Required skills:

- knowledge about measurements and protocols related to Web browsing (HTTP, DNS, TCP...)

References:

[1] J. Charzinski: Web Performance in Practice -- Why We are Waiting. AE Vol. 55 No. 1, Jan. 2001, pp. 37-45

UiO2 – Performance evaluation of Mercury for managing network measurements in real world

Responsible: Matti Siekkinen, University of Oslo, siekkine@ifi.uio.no

Overview

Mercury [1] is a system for supporting multi-attribute range-based searches in a scalable way. The objective of this assignment is to evaluate how Mercury performs when used in a real world system. The idea is that different aspects of the system are evaluated when running several Mercury nodes in Planetlab. These aspects include CPU utilization, bandwidth utilization (queries vs. data records), query response time, data insertion delay etc.

Tasks to do:

- deploy Mercury on Planetlab
- establish measurement infrastructure:
    - Mercury already collects lots of different logs but more may be necessary
    - traffic measurements (tcpdump and friends)
- define experiments
    - data records to store
    - queries to issue
- run experiments and analyze results

Required skills:

- C/C++ skills, knowledge about tcpdump/wireshark is useful

References:

[1] A. R. Bharambe, M. Agrawal, and S. Seshan, "Mercury: supporting scalable multi-attribute range queries", In SIGCOMM 2004.

UiO3 – Extending a DSMS benchmark

Responsible: Jarle Søberg, University of Oslo, jarleso@ifi.uio.no

Overview

We are currently working on evaluating several data stream management systems (DSMSs) for network monitoring. We have developed a benchmark, which manages to handle all different DSMSs by defining a simple I/O and performing "black box" testing of three DSMSs. The benchmark works by generating network traffic, which is then obtained by a machine that listens to all the traffic. We use four metrics to evaluate the DSMSs: accuracy, response time, processing rate, and CPU and memory consumption.

Task

The assignment is divided into two different tasks:

1. Extend the benchmark with new queries that resemble interesting tasks in network monitoring. It is important to argue why these queries are used. Also consider e.g. window lengths and types.
2. Test the new queries, as well as the old ones, in TelegraphCQ, STREAM, Borealis, and, if time, in ESPER.

Required skills:

- Some knowledge in Perl, C, and declarative languages used in DSMSs and DBMSs (like SQL).

References:

[1] Thomas Bernhardt and Alexandre Vasseur, "Esper: Event Stream Processing and Correlation", http://www.onjava.com/pub/a/onjava/2007/03/07/esper-event-stream-processing-and-correlation.html

UiO4 – Online analysis of medical sensor data with the Esper Event Stream Processing System

Responsible: Morten Lindberg, University of Oslo, mglindeb@ifi.uio.no

Overview

Esper Event Stream and Complex Event Processing for Java http://www.espertech.com/ let users perform continuous queries, aggregations and joins on live data streams. Meaning the system falls into the DSMS (Data Stream Management System) category which INF5090 students should be familiar with.

Recent work at the Interventional Centre (IVC) at Rikshospitalet involves connecting patient biomedical sensors through a wireless network. The people at IVC have mainly focused on the low-level networking issues, rather than high-level data analysis and management issues. IVC has agreed to give us data from real situations, meaning we can perform playback of live data from medical operations. The data is collected in real-time in a Java-based application. The idea for this assignment is to perform real-time analysis of the data streams derived from the biomedical sensors, and perform online query processing on them with the use of Esper.

Task

Current requirements of the continuous queries are not decided yet. Ideally queries performing both filtering of unnecessary data, and monitoring and identification of critical health conditions should be presented, although only to the extent that it is possible without severe understanding of medical meaning of the data. The students should try to identify the usefulness (e.g., limitations, performance, and how to express queries) of such a system in this domain.

Required skills:

- Java and SQL skills, but most of all the ability to work within an unfamiliar domain (like DSMS / Event Stream Processing).

References:

[1] Thomas Bernhardt and Alexandre Vasseur, “Esper: Event Stream Processing and Correlation”, http://www.onjava.com/pub/a/onjava/2007/03/07/esper-event-stream-processing-and-correlation.html

UiO5 – Using OLSR Beacons (and Broadcasts) for Higher Layer protocols in MANETs

Responsible: Katrine Stemland Skjelsvik, University of Oslo, katrins@ifi.uio.no

Overview

Bandwidth is a scarce resource in MANETs. However, MANET routing protocols as well as many other higher layer protocols and applications for MANETs need to send regularly beacons, hear beats etc. The amount data encapsulated in beacons, i.e., the payload of the corresponding packets, is rather low, but the message itself represents non neglectible overhead. This is especially severe if there is not only the routing protocol, but also several other higher-layer protocols that send regularly from each node their own beacons.

Tasks

In order to reduce this overhead, we aim to provide a generic beacon service on each node, which builds on the OLSR beacon. The basic idea is to piggy-pack all different beacons at heartbeats into a single (extended) OLSR beacon. To achieve this goal, the following tasks have to be performed:

- A plug-in needs to be develop that intercepts the beacons and adds at the sender side higher level beacons as payload and removes them at the receiving side.
- A corresponding packet format needs to be defined to allow the arbitary beacons to be piggy-packet as payload.
- A programming interface needs to be designed that allows arbitary protocols to piggy-pack their beacon, and to receive the beacons from other nodes.
- The solution needs to be tested and evaluated with respect to backward compatibility.

Required skills:

- Java language C is requiered and some experience in using Linux is of advantage.

References:

[1]http://www.olsr.org

[2]NEMAN, http://www.ifi.uio.no/forskning/grupper/dmms/papers/138.pdf