Institut für Parallele und Verteilte Systeme
Abteilung Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D - 70569 Stuttgart

Master Thesis Nr. 3667

Optimal Routing for Networked Control Systems using OpenFlow

Ha Duy, Bach

Studiengang:	Infotech
Prüfer:	Prof. Dr. rer. nat.Kurt Rothermel
Betreuer:	Dipl. -Ing. Ben W. Carabelli
begonnen am:	02.06.2014
beendet am:	01.03.2015
CR-Klassifikation:	

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben.
Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen
Werken übernommene Aussagen als solche gekennzeichnet.
Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens.
Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht.
Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Unterschrift:

Ort, Datum:

Declaration

I hereby declare that the work presented in this thesis is entirely my own.
I did not use any other sources and references that the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations.
Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before.
The electronic copy is consistent with all submitted copies.

Signature:

Place, Date:

Abstract

Networked Control System (NCS), which implements a feedback control loop over a communication network connecting sensors, actuators and controller, has gained a significance demand. However, in many cases, the infrastructure is not previously designed for dedicated control network because of the cost or because of the technology vision. Therefore, NCS should utilize the popular IP networks, which often do not provide any QoS provision. This is a challenge as NCS are sensitive toward delay and loss. In this thesis, I want to revisit a previous study on NCS, as well as to look at the measurement of delay in previous literature and current market devices. After that, I will point out the challenges of implementing NCS services of the previous study in real switches supporting SDN technology. Finally, I propose a solution on the most popular platform of SDN technology and demonstrate my evaluation.

Content

List of figures

List of tables

List of equations

1. Introduction

Networked Control System (NCS) has gradually become an important part in the industry. Implementing a feedback control loop connecting various distributed sensors, actuators and controller, NCS provides an automatic responsive system to satisfy specific demand. However, NCS might face with various challenges from the implementing environment.

Take a very specific example of a hospital, which wants to upgrade the heating system to an automatic temperature control system. This temperature control system will automatically adjust the temperature of every room inside the hospital according to a sufficient plan. However, they cannot have the resource and they cannot disturb so many patients. Therefore, the new NCS should be implemented on the available Ethernet network, which has already been designed with enough bandwidth. Since the network is mainly used for patients and staffs to access the internet and some internal website, it provides no QoS control service.

From the example, we can see the challenges of implementing a NCS. In the first place, it will not always be possible to provide a dedicated channel for the NCS. Therefore, NCS should be implemented on a readily available network, particularly, the popular Ethernet network. Secondly, NCS are vulnerable to delay and loss. It should be especially hard to build a NCS on top of a packet-switched network for example. With this in mind, and with the popularity of the Ethernet network, we can already narrow down that this network will be the environment in which NCS is going to run most often. However, the Ethernet network may not always provide QoS services like IntServ, which guarantees the delay bounds, or even DiffServ, which provides low-latency to critical network traffic. NCS might need to work on a best-effort IP network.

This thesis will revisit a previous study - “A Network Abstraction for Control Systems” (1), in which the authors propose a probabilistic model of NCS, which can account for certain degree of delay and loss on a best-effort network. This is further realized in NCS services and an optimal routing mechanism utilizing the state of the art software-defined networking (SDN) technology. The implementation was made in a simulation environment with sufficient control over many network characteristics, thus provide the first insight of utilizing modern SDN technology to solve optimization problem of the increasingly on demand NCS.

Recently, there is a fast standardization of SDN toward OpenDaylight project, with the backup of all big network companies in the West. Along with it is the rapid development of virtual switch system such as OpenvSwitch that support SDN, support virtualization as well as being ported to switching chipsets. This brings us to a new environment, which should be studied and trialled up on. However, moving from an abstract study and simulation implementation to market proven products, forming a new environment, will bring a lot of challenges and uncontrollable factors. I hope the Thesis can give some valuable insights to the problems and solutions of this interesting change.

2. Previous study revisit

In this part, we will quickly revisit a previous study of NCS, namely “A Network Abstraction for Control Systems”. We will look at the introduced system and have a brief understanding of how NCS work in the network.

2.1. System models

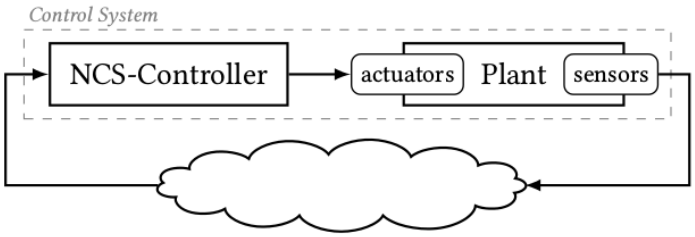

Figure 1: System model

(1)

The system in (1) have in total five components: the plant, sensor, actuator, NCS-controller and communication network. The plant is the physical object that is controlled by the system. Sensors measure the status of the plant and transfer the information over the communication network. The controller will receive the information from sensors and make calculation base on these information. After that, the controller will send the required action to the actuators for execution.

2.2. Network architecture

Following is the network architecture of the networked control service:

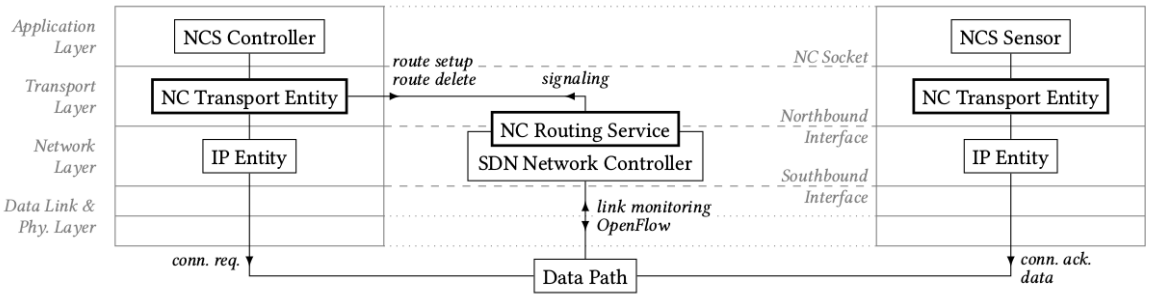

Figure 2: Network architecture

(1)

In figure 2, the routing service is located between Network layer and Transport layer. It acts as a socket service to deliver routing information and to signal changes in the network. The

implementation require a physical or virtual SDN controller and the support of OpenFlow on switches.

2.4. NC routing service

Following is the brief view of the routing algorithm:

Require: $G = (V, E, \mathcal{P}, \mathcal{L})$ – network graph
$p_{min}(\cdot)$ – minimum arrival probability (provided by application)
$v_s, v_c \in V$ – sensor and controller node
Ensure: σ_{opt} solves (11)

1: **function** OPTIMALROUTE($G, p_{min}(\cdot), v_s, v_c$)
2: **for all** nodes $v_n \in V$ **do**
3: $R(v_n) \leftarrow \emptyset$
4: **end for**
5: $R(v_c) \leftarrow \{v_c\}$ ▷ *initialize path set on v_c*
6: $M \leftarrow \{v_c\}$ ▷ *mark v_c for relaxation*
7: **repeat**
8: **for all** $v_n \in M$ **do**
9: **for all** v_i with $(v_i, v_n) \in E$ **do**
10: RELAX(v_i, v_n)
11: **end for**
12: $M \leftarrow M \setminus \{v_n\}$ ▷ *unmark node v_n*
13: **end for**
14: **until** $M = \emptyset$
15: $\sigma_{opt} \leftarrow \arg\min_{\sigma \in R(v_s)} \left(\mathrm{E}_\tau(\sigma)/T_{max}(\sigma)\right)$
16: **return** σ_{opt}
17: **end function**

18: **function** RELAX(v_i, v_j)
19: **for all** $\sigma \in R(v_j)$ **do**
20: $\hat{\sigma} \leftarrow v_i\sigma$ ▷ *new candidate path*
21: **if** $v_i \in \sigma$ **or** $\hat{\sigma} \in R(v_i)$ ▷ *loop/duplicate*
22: **continue**
23: calculate $\mathrm{E}_\tau(\hat{\sigma})$, $\mathcal{P}(\hat{\sigma})$, and $\mathcal{L}(\hat{\sigma}, t)$
24: **if** $\mathcal{P}(\hat{\sigma})\mathcal{L}(\hat{\sigma}, t) < p_{min}(t)\ \ \forall_{t>T_{min}}$ ▷ *$\hat{\sigma}$ unfeas.*
25: **continue**
26: **if** $\exists\, \sigma \in R(v_i) : \sigma \gg \hat{\sigma}$ ▷ *$\hat{\sigma}$ dominated*
27: **continue**
28: $R(v_i) \leftarrow R(v_i) \setminus \{\sigma \mid \hat{\sigma} \gg \sigma\}$ ▷ *purge $R(v_i)$*
29: $R(v_i) \leftarrow R(v_i) \cup \{\hat{\sigma}\}$ ▷ *add $\hat{\sigma}$ to $R(v_i)$*
30: $M \leftarrow M \cup \{v_i\}$ ▷ *mark v_i for relaxation*
31: **end for**
32: **end function**

Figure 3: The NC routing service algorithm

(1)

More information on the algorithm such as network theory, equations, abbreviations, etc. can be found inside the paper (1).

3. Implementation challenges

Whether in virtual or real network devices, the support for SDN technology is increasing rapidly. The implementing network devices should support SDN technology, and specifically the OpenFlow protocol. This protocol will provide communication between the device and the SDN controller, which is supposed to manage and customize the routing of all network devices. Moreover, the technology also provides us with a lot of information on the network, such as detail name, IP address, or even the map of the whole network. A dedicated statistic module is also available inside SDN controllers like OpenDaylight. However, this information is not enough for deeper study. With the current support of SDN controllers, even the well-known OpenDaylight does not provide some specific information we need for our implementation of the algorithm in (1).

With the use of simulation network, we have control over almost everything in the network, from the switches, the link characteristics, and even the queue. Moving to real/virtual switches such as OpenvSwitch, we further leave back the simulation environment we have. This is what the thesis intend to do, reach out for potential real environment. However, many measuring information and characteristic control have been lost in this process, specifically, the two important characteristics: delay and arrival probability. These two characteristics are of great important inside our algorithm and we have to find a way to compensate for this lack of control.

Real delay measurement has been left with not much concern in the field of network management. Many researches are using simulation network in which they can directly control or obtain the delay time, therefore not many papers have focused on the actual measurement of delay. A notable study on one way delay measurement is “One-way Delay Measurement and Characterization” (2). However, in this study, the authors have to use the dedicated network European Traffic Observatory Measurement Infrastructure (ETOMIC) and a specially enhanced network interface card to measure and analyse the delay.

Meanwhile, the same situation on delay measurement happens with the products on the market. It is very surprise that big company such as Cisco still use two-way delay measurement, which is actually the round-trip time, as the delay information for their VOIP system. Other software players introduce the average delay of multiple links between to dedicated devices, the client delay and server delay, etc. but the one way delay

measurement, which is the most important delay information, is not addressed. Of course the one way delay measurement is introduced in some less popular network players such as Accedian Networks Inc., and it requires a dedicated network of this brand to fully work. The delay in this scenario is measured by time-stamping test packets sent out on the network. However, we often see Cisco devices flood into a company and create a dedicated network rather than these unpopular players.

Time synchronization is also a problem to look at since the one way delay measurement requires the time from two different devices. Therefore, the network should be implemented with a good time synchronization technique. As of now, most network devices use time synchronization system such as NTP or SNTP. However, we cannot guarantee the time synchronization in some network overload or network changes. In our current state, we do not have the ability to take into account these situations. Meanwhile, some one-way delay measurement papers such as (3) actually mention an expensive way to make time synchronization more reliable. With the dedicated network of dedicated devices, they are using GPS time synchronization attached to each device to make the time synchronized in one of the most reliable way. As this said, it is not going to happen easily in our implementation environment, where the client do not want to build a dedicated network or to invest too much into the matter.

Let us go into more detail about these challenges with specific literature, papers and market products.

3.1. Available alternatives for measuring delay on literature and market

First, we pay a visit to some papers and documents that we have been talking about.

3.1.1. Two-way delay measurement

The first paper is from K'evin Phemius and Mathieu Bouet, named: "Monitoring latency with OpenFlow" (4). This is rather an interesting paper because they propose a mechanism: *"measure link latencies from an OpenFlow controller with high accuracy and a low footprint"*. This shows that people have been moving to new technology in order to solve the problem of latency. Moreover, the specific approach of this paper toward using round

trip time in sufficient way is admittedly the same as my approach when I first begin the Thesis.

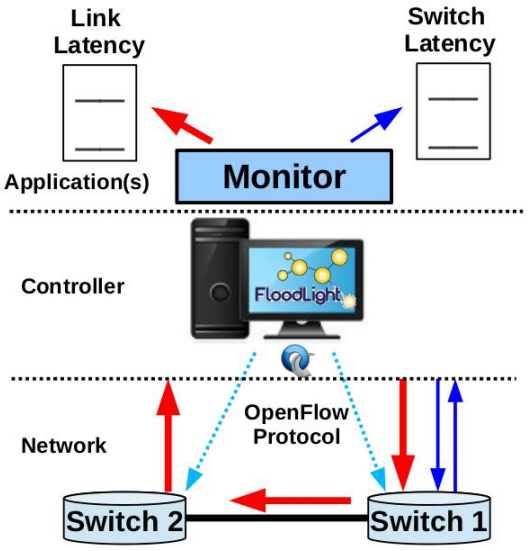

Figure 4: Monitoring mechanism

(4)

Here they measure the latency by sending out probe messages and measure the time different. According to their measurement, they will achieve the link's latency by the following equation:

$$Latency(s_1, s_2) = T_{total} - \frac{T_{s_1}}{2} - \frac{T_{s_2}}{2} \text{ - C}$$

Equation 1: Link's latency

(4)

Here the value is described as:

- T total: The total time the packet come from the controller to the first switch, to the second switch and come back to the controller.
- T s1: The round trip time between the controller and one switch.
- T s2: The round trip time between the controller and the other switch.

This way of measurement is very interesting because it is not completely round trip time. In an environment that the controller has not much load, the measurement should be very close to the exact delay.

As for another example with the big company, we can take a look at “Measuring Delay, Jitter, and Packet Loss with Cisco IOS SAA and RTTMON” (5) from Cisco. Inside this document, Cisco is going to define delay, jitter, and packet loss, as well as demonstrate it over their VoIP products.

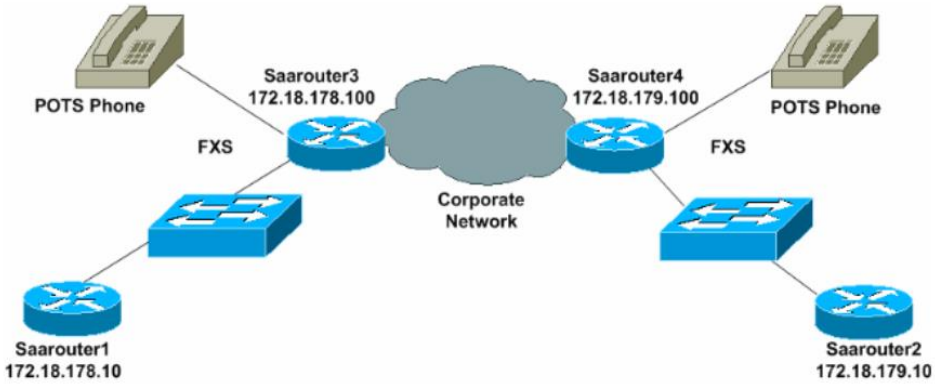

Figure 5: Delay and Jitter Probe Deployment example by Cisco

(5)

As far as the characteristic of VoIP goes, it is very vulnerable to jitter, loss and delay. Delay exceeding 150ms is unacceptable for a call. However, Cisco admitted right away in their document that:

> *“One−way delay calculations require expensive sophisticated test gear and are beyond the budget and expertise of most enterprise customers. However, measuring round−trip delay is easier and requires less expensive equipment. To get a general measurement of one−way delay, measure round−trip delay and divide the result by two.”* (5)

But the problem may lie on only one direction of the call while the average delay is still acceptable. Clearly, even big player like Cisco does not want to invest too much on delay problem. However, I won’t criticize them because their deployment environment of VoIP most likely end up in private networks with good provisioning or cisco dedicated network, and what they solve when deploying a telephony system may already be: the possibility of having a bad call should be very low. This can be achieved by provisioning a dedicated network with traffic never exceed 70-80% of the full capacity and processing power. This way of provisioning for enterprise customers will last the network for at least 5 to 10 years. This is also the reason why Cisco can boldly admit that they only use the round trip time to measure the delay.

Having some interesting visit over the papers of the two way delay measurement, we also want to look at the development in one way delay measurement literature.

3.1.2. One-way delay measurement

First, we visit “One-way Delay Measurement and Characterization” from Ana Hernandez and Eduardo Magaña (2). In this paper, they emphasis on the precision of one way delay measurement by using the European Traffic Observatory Measurement InfrastruCture (ETOMIC). The European Traffic Observatory is a European Union VI Framework Program sponsored effort, within the Integrated Project EVERGROW, that aims at providing a pan European traffic measurement infrastructure with high-precision, GPS-synchronized monitoring nodes. Up to this date, there are 16 nodes distributed in Europe.

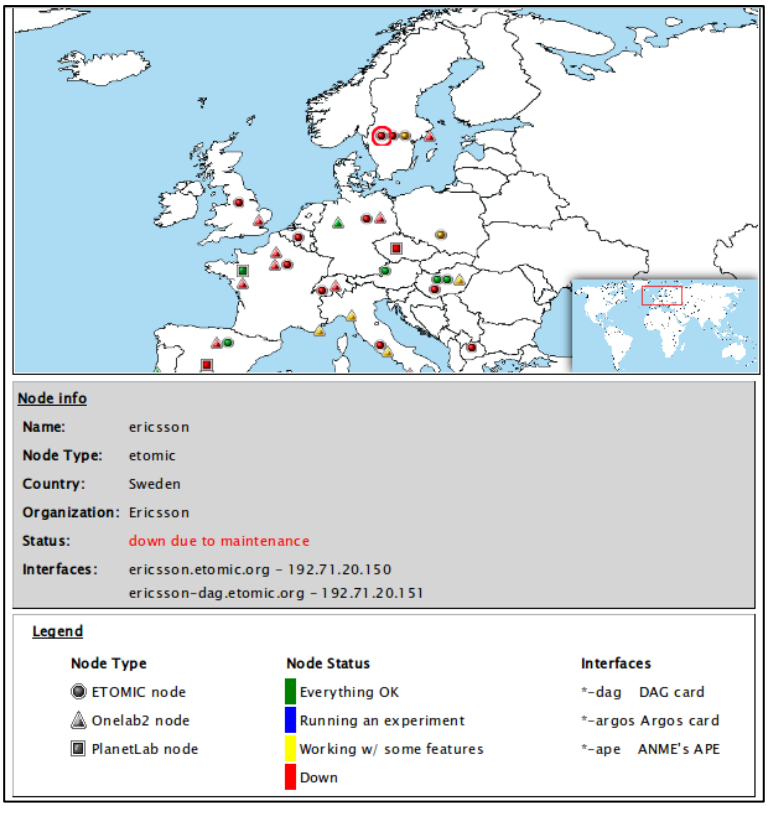

Figure 6: Overview of ETOMIC nodes on a web tool

(2)

These ETOMIC nodes use a rather advanced network card (DAG Endace 3.6GE) capable of adding time stamp. This allows them to have a high precision because it avoids the problem

of having the packets going into the application layer to receive the time stamp, which clearly reduce the precision of the measurement. According to the paper, ETOMIC allows them to obtain one-way delay measurements with accuracy in the order of hundreds of nanoseconds. This is quite a significant achievement.

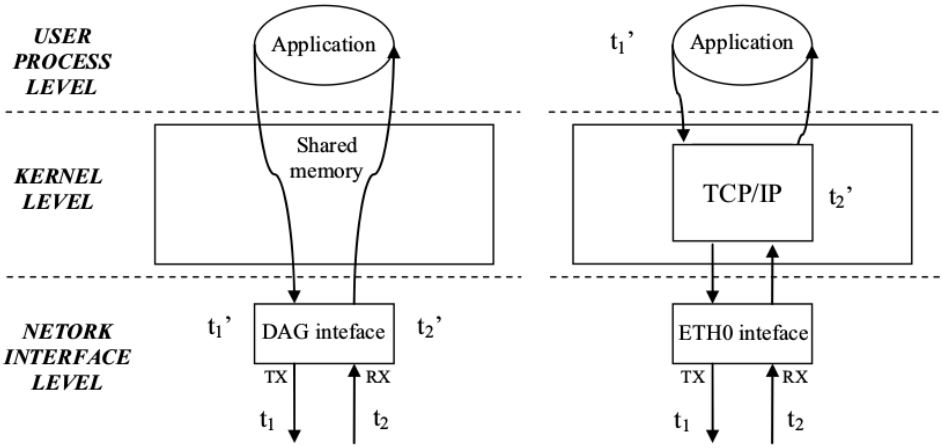

Figure 7: Time stamping on network card and time stamping at application layer

(2)

However good these dedicated networks of specially configured nodes are, we should also look back at our problem, which does not provide us with these wonderful environments. But we should also admit that this might be one of the best tools to practice one way delay measurement out there.

Another look on one way delay measurement is presented in a white paper “One-Way Delay Measurement Techniques” (3) from Accedian Networks. In this paper, we realize two most important factors to look at when one want to measure one way delay.

The first problem is clock synchronization accuracy. According to this white paper, since all the precision of our measurement are based on time, clock synchronization with NTP and PTP cannot satisfy the need. The design of traditional central sync NTP/PTP server providing time information for the network through propagation may have some limit that affect the measurement accuracy. Therefore, they propose their “patent-pending enhanced NTP statistical sync algorithm that greatly reduces the effects of high frequency delay asymmetries caused by network congestion”. According to the paper, the units can provide

one-way delay and jitter measurements with <20 µs accuracy, and 1 µs resolution. This is of course significantly more accurate than the original NTP infrastructure.

Another approach the paper proposed for clock synchronization is of course using GPS enabled devices on the network. This approach save the need for NTP algorithm or advanced NTP algorithm, but is also not easy to implement. A dedicated network with specially configured devices is what the implementer wants, but rarely what the customer wants.

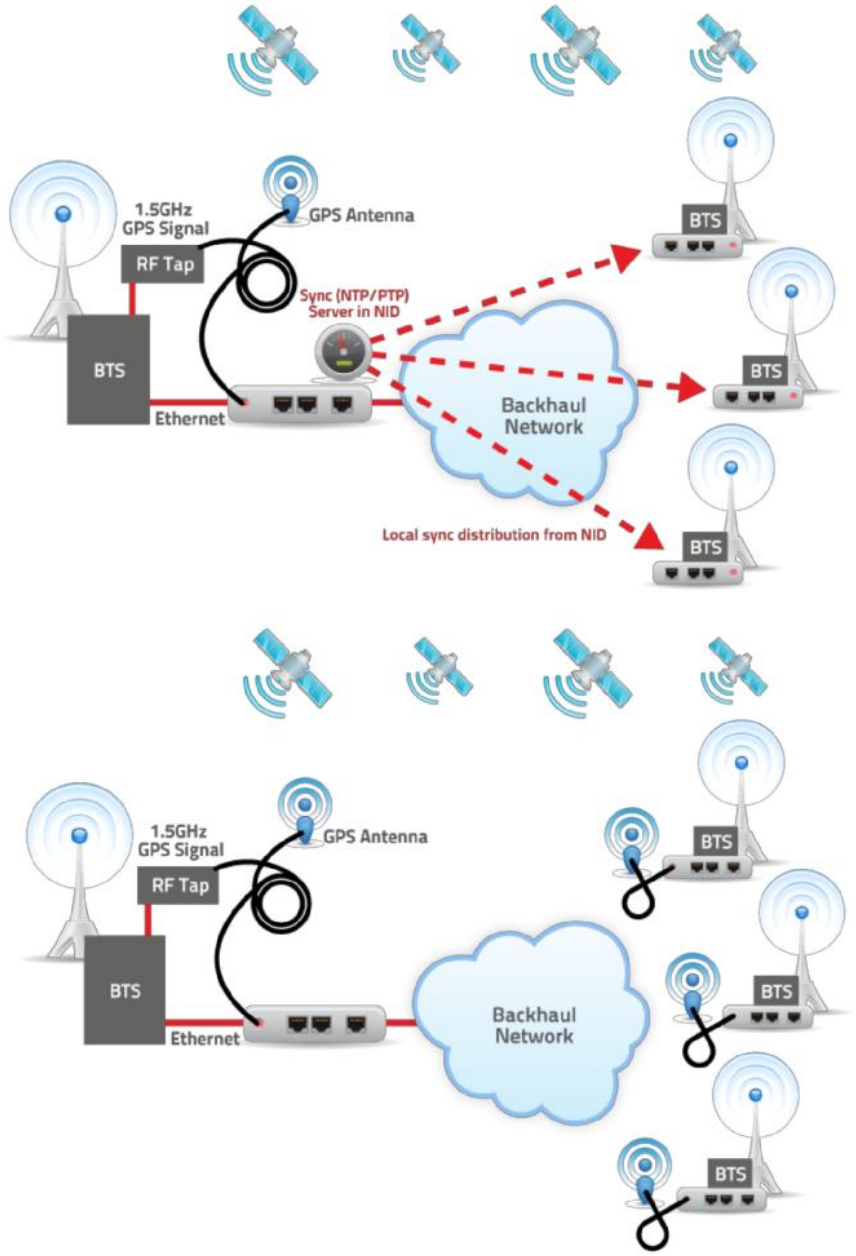

Figure 8: Using NTP server as a GPS enabled devices and using all GPS enabled devices (3)

The second problem that the paper points out is the architecture of the devices may affect the delay measurement itself. The time-stamping and packet processing at what point inside the architecture will significantly affect the precision of the measurement. The Accedian Networks's unit provide time stamping at the hardware level, which is of course the most ideal one. That is why they can achieve the precision of microseconds.

As we can see from the two papers we visited, one way delay measurement has become more or less a very heavy and expensive task if one want to solve the entire problem around it. That is why from the perspective of our NCS system and our environment, it is much harder to overcome. We cannot easily convince the customer to build a dedicated network or invest on specifically configured devices.

Now, let's look at delay measurement from another perspective, which is protocols and software systems that try to tackle on delay measurement.

3.1.3. Protocols

Since we are measuring the delay of a packet of a flow, we are going to talk more about flow and flow monitoring here. The whole SDN technology is based on flow, so we can't negate this part out when we talk about using it in our measurement. As we have said earlier, the statistics that SDN controllers offer are not enough for our algorithm. Therefore, we look out for flow monitoring mechanism that can provide us with the information we need.

While each manufacturers support alternative flow technologies including; Juniper (Jflow); 3Com/HP, Dell and Netgear (Sflow); Huawei (NetStream); Alcatel-Lucent (Cflow); and Ericsson (Rflow); Cisco (NetFlow), there are two most famous protocols that people often use when monitoring flow, namely: Sflow and NetFlow/IPFIX. While Sflow seems to be supported by more manufactures, NetFlow is a Cisco protocol and IPFIX is just a IETF standardization of NetFlow version 9. Here, we will look deeply at the two protocols, Sflow and NetFlow.

Sflow is actually a name taken from "sampled flow". It works at the Layer 2 of OSI model and export packet at this layer with a counter interface. A Sflow enabled switch will send information to the collector, which can be the controller itself or an independent entity inside the network. Then, this data normally are analysed by a data analyser. Sflow's

sampling mechanism can be divided on packet-based sampling as well as time-based sampling. Since sampled packets are sent to the collector, Sflow may create additional hit on the traffic of the network. Therefore, one need to consider the available bandwidth and the timing to avoid an overload on the network.

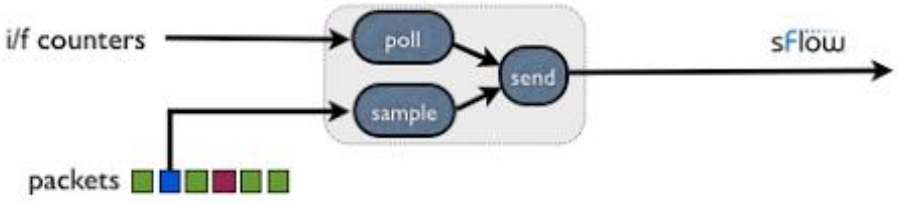

Figure 9: How Sflow sampling works

(6)

Since Sflow's mechanism is based on sampling, it is particularly good with large flow and large traffic, and it can achieve a certain degree of scalability. To achieve scalability as well as accuracy, the sampling rate of Sflow should vary according to the link capacity.

Link Speed	Sampling Rate
10Mb/s	1 in 200
100Mb/s	1 in 500
1Gb/s	1 in 1000
10Gb/s	1 in 2000

Table 1: Sampling rate should vary according to the link capacity

(6)

Of course this should be applied according to specific environment. For example, we may not need a very high sampling rate if the traffic in such environment is unusually high all the time. Having a look at the datagram of Sflow, we are not seeing a very clear table of field and definition. What we see is many structs of different functionality, which is the programming representation of different packets and information sent to the controller. This shows that Sflow is flexible to develop and add some additional information in the future if they want.

```
/* Packet IP version 4 data */

struct sampled_ipv4 {
   unsigned int length;     /* The length of the IP packet excluding
                               lower layer encapsulations */
   unsigned int protocol;   /* IP Protocol type
                               (for example, TCP = 6, UDP = 17) */
   ip_v4 src_ip;            /* Source IP Address */
   ip_v4 dst_ip;            /* Destination IP Address */
   unsigned int src_port;   /* TCP/UDP source port number or
                               equivalent */
   unsigned int dst_port;   /* TCP/UDP destination port number or
                               equivalent */
   unsigned int tcp_flags;  /* TCP flags */
   unsigned int tos;        /* IP type of service */
}
```

Figure 10: Example of a struct in Sflow datagram version 5

(7)

Talking about Sflow at the data analyser's perspective, the advantage of Sflow is in its mechanism to send the whole packet to the collector. This can lead to up-to layer 7 monitoring if one desire. However, Sflow also have a big disadvantage, because of its sampling mechanism, the data analyser don't have all the information to graph exactly the used bandwidth for example.

In summary, Sflow use "sampling" technology and send the whole packet or a truncated packet to the collector. Up to date, no stamp and not so much information is added to the packet, Sflow just send RAW or truncated RAW packet to the collector. Sflow was supported on the market by a large amount of vendors.

The other protocol that we want to talk about is NetFlow. This is a protocol introduced on Cisco routers providing the ability to aggregate network traffic information into different flow information. Unlike Sflow, NetFlow does not work on sampling mechanism, which means that once activated, NetFlow will monitor every packets that when through the configured interfaces. NetFlow is based on identifying packet flows for ingress IP packets, which is at the layer 3 in the OSI model. The difference of NetFlow is that after intercept the packet, it takes out specific information needed for the monitor and aggregates this information with its current data. Therefore, NetFlow will take a little bit processing power from the switch. Periodically, NetFlow enabled switch will send out its aggregate information to a collector in the network. This information will again, go into a data analyser.

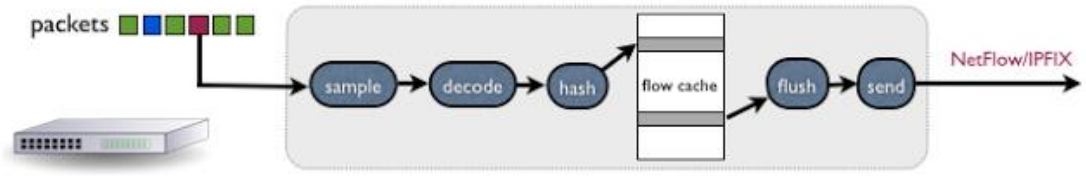

Figure 11: How NetFlow handle ingress packets

(6)

Since NetFlow consumes additional memory and CPU resources, it is important to understand the resources required on the device before we enable NetFlow. Especially we shouldn't really enable NetFlow on switch or router working in place that have high possibility of receiving sudden large traffic or continually large traffic that may, in some configuration, exhaust most of the CPU and memory resources.

Figure 12: NetFlow graph in Solarwinds's "NetFlow Traffic Analyser"

(8)

Some might say that this is a disadvantage of NetFlow, but some might yet say, this is an advantage. From the point of the protocol, it shows clearly that NetFlow will only stop at

Layer 3 of the OSI model as Cisco intended, leaving the need for monitoring at layer 4, 5, 6, and 7 to dedicated network analysers. From the point of collecting data, and giving it to the analyser, NetFlow did a great job of giving all necessary information for graph processing, no more, no less. This leads to the favour of NetFlow data analyser products on the market. Although NetFlow is not supported by many manufactures, Cisco customer can be quite satisfied with the result from the data analyser.

In giving more advanced configuration and a new face to its own protocol, Cisco later release a Flexible NetFlow. In this new version, there is an option to change NetFlow from interpreting every packet to sampling with a certain sampling rate to reduce CPU overhead. This is clearly a step to cope with the need of running Flexible NetFlow on cheaper devices.

NetFlow in Cisco switches have a clear configuration guide with a lot of timing options for the flows, the aggregation of data and the sending of information to collector. It is different from other flow monitoring protocols, for which you may need to give command in a Linux-like manner to activate. The datagram of NetFlow is also very clear and easy to understand, it consists of tables of field and definition with clear explanation.

Bytes	Contents	Description
0-1	version	NetFlow export format version number
2-3	count	Number of flows exported in this packet (1-30)
4-7	SysUptime	Current time in milliseconds since the export device booted
8-11	unix_secs	Current count of seconds since 0000 UTC 1970
12-15	unix_nsecs	Residual nanoseconds since 0000 UTC 1970
16-19	flow_sequence	Sequence counter of total flows seen
20	engine_type	Type of flow-switching engine
21	engine_id	Slot number of the flow-switching engine
22-23	sampling_interval	First two bits hold the sampling mode; remaining 14 bits hold value of sampling interval

Table 2: The NetFlow version 5 packet header

(9)

In summary, NetFlow interprets every ingress packets and aggregate flow information of the packet to send to the collector. NetFlow does utilize CPU and memory resources in the

process. The variety of configuration options and the clearance of the datagram bring ease to people who want to use them. NetFlow version 5 and NetFlow version 9 are often supported in an adequate amount of vendors. The popular of NetFlow version 5 is still undeniable as time passes.

3.1.4. Software products

As for software products on the market, which are capable of measuring delay, many need the dedicated system that we have shown earlier. This means that these tools go tightly with these devices and cannot work without them. The other software products also try to measure delay in various ways possible. Most resort to two way delay measurement, some rely on the protocols that we previously study, some present interesting mechanism. Anyway, in this section, we won't go into the dedicated system, as they have already been touched in section 3.1.2. What we want to see is: How have the other products which do not require dedicated devices been doing? And how has the open source network monitoring been doing?

One interesting whitepaper "Measuring Latency Using NetFlow" (10) from Luca Deri of ntop.org and Michael Patterson of Plixer.com shows how they use their own probe on top of NetFlow to determine some network delays. This is not really going straight with our path, but the way they use their probe and the way they measure give suggestive and interesting idea for our study. Furthermore, although one will be charged for the solution, Nprobe is actually opensource software, which may prove useful when one want to go deeper into the code and include more functionality.

To understand the whitepaper, firstly we must understand how Nprobe works on NetFlow. Why the authors clearly use most of the functions from Nprobe but still call this whitepaper "Measuring Latency Using NetFlow" (10). Nprobe basically sit at NetFlow agent side, and take all the aggregate information that NetFlow have to build its own database and send to the collector. However, since Nprobe is not only restricted to NetFlow, it can collect more information or setup more things of its own. Nprobe has three operating mode, of which two are related to NetFlow. The NetFlow collector in this situation is of course also software that go along with Nprobe called Ntop.

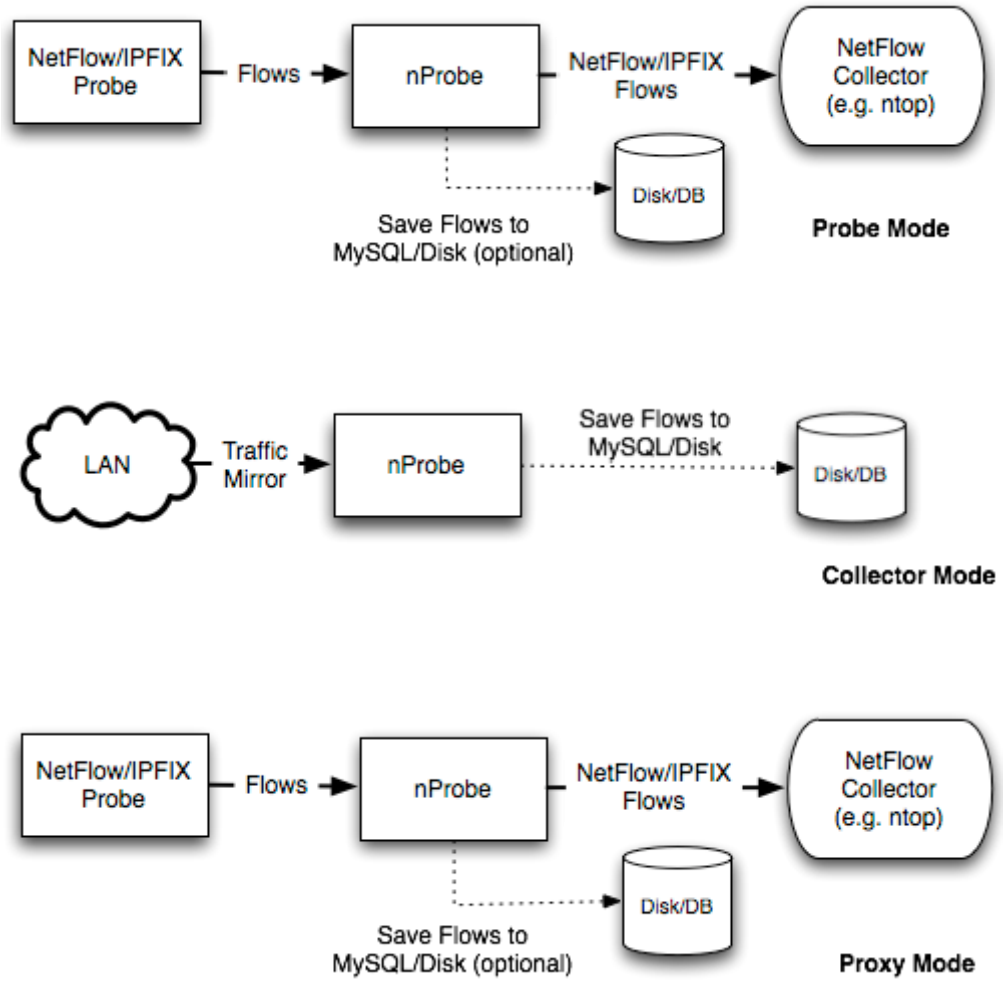

Figure 13: "Nprobe has three operating mode, of which two are related to NetFlow"

(11)

The only different between Nprobe probe mode and proxy mode is that, in proxy mode, you can convert the NetFlow version of the information that NetFlow send out. Of course one can lose some information when converting between different NetFlow versions.

Now that we have understood the mechanism of Nprobe, let's look at how Nprobe work with its application delay measurement. By observing the TCP flag at the beginning of a

transaction (three ways handshake), Nprobe, at the middle of the client and server, obtains two important timing value: SERVER_NW_DELAY and CLIENT_NW_DELAY.

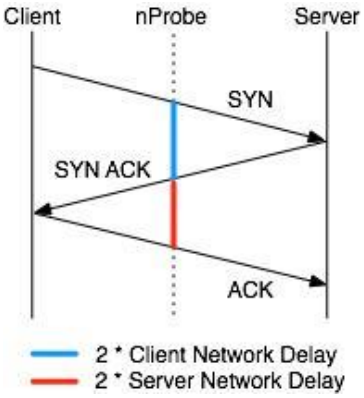

Figure 14: SERVER_NW_DELAY (Server network delay) and CLIENT_NW_DELAY (Client network delay)

(10)

This is due to the fact that from RFC1323, TCP is redefined with very useful timestamps fields. TCP timestamp are not guaranteed to be aligned with the clock and can start at some random value. It seems like the implementer has got some tweak so that the time stamp is aligned with the clock. Also, in the calculation to achieve the CLIENT_NW_DELAY and SERVER_NW_DELAY, the authors have already divided the result by two in the end, revealing that this is a two way delay measurement. As for the APPL_LATENCY, with the assumption that Nprobe is placed very near the server, the paper take the time difference between a TCP request and a TCP reply that is interpreted by Nprobe to be the APPL_LATENCY. It is critical to ask for the accuracy of this calculation, but the paper didn't show any practical design and provide little information on this third calculation, so we may pass on it.

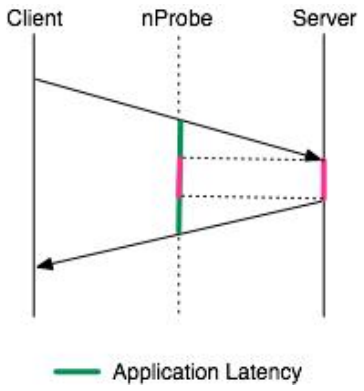

Figure 15: APPL_LATENCY (Application latency) seems to be a hard topic

(10)

Clearly, the whitepaper (10) shows us some visibility of latency on the network by utilizing three important things: NetFlow, Nprobe and TCP timestamp. With these ability, Scrutinizer, which is the company's network monitoring tool, can show the administrator valuable information in demanding time. The method used is two way delay calculation to calculate some delay timing value. And it leave us with a not very satisfied feeling when beginning with some great talk but end up not carefully solving the application latency matter.

However, the significant part in this matter is that one can utilize a probe to get NetFlow information. The combination of this information with the functionality of the probe may bring out some interesting features.

In the opensoure software, the most famous tool dedicated for delay monitoring should be SmokePing. SmokePing use live graph to display the network delay from the monitoring point to any places inside the network. Moreover, SmokePing also offer a wide range of latency measurement plugins in the form of probes, which is also the central working mechanism of SmokePing. SmokePing also offer some interesting features, such as: distributed measurement and alerting system. It is to be noted that having gone through the SmokePing probes, I can tell that these delay measurement are two way measurement, which is actually half of the round trip time.

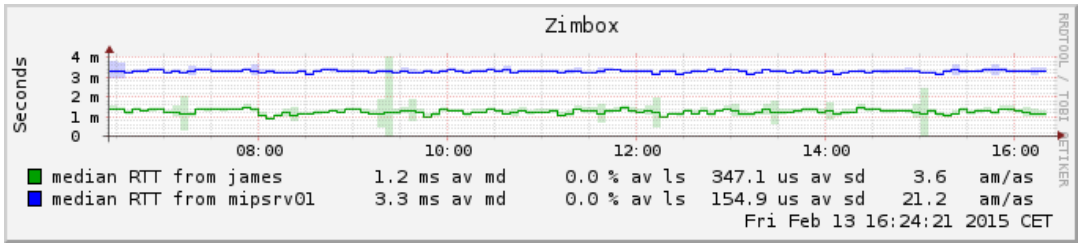

Figure 16: SmokePing use live graph to display the network delay

(12)

So how exactly does SmokePing work? As I have mentioned, SmokePing work with an interesting probe mechanism. There are always two steps in the configuration of SmokePing. The first step is probe configuration. Here, one has to specify the probe name and additional variables along with this probe. There are many types of probe, from monitoring DNS servers to monitoring SSH protocol; even monitoring a HTTP link is possible. This is part of a sample of probe configuration:

```
*** Probes ***
+ FPing
binary = /usr/bin/fping
packetsize = 1000

+ DNS
binary = /usr/bin/dig
lookup = name.example
pings = 5
step = 180

+ EchoPingHttp
pings = 5
url = /test-url
```

Figure 17: SmokePing probe configuration

(12)

The second step is target configuration. Inside target configuration, one can specify the site, the host, the DNS, anything that you want the probe to monitor. Each target will of course have the probe name that should be associated with it. Here we can also see a part of the configuration file:

```
*** Targets ***
probe = FPing
menu = Top
title = Network Latency Grapher
remark = Welcome to this SmokePing website.

+ network
menu = Net latency
title = Network latency (ICMP pings)

++ myhost1
host = myhost1.example
++ myhost2
host = myhost2.example

+ services
menu = Service latency
title = Service latency (DNS, HTTP)
```

Figure 18: SmokePing target configuration

(12)

These simple configuration and various probes make SmokePing robust, and getting enough feature to satisfy people. Although SmokePing concentrates on a very small part of network monitoring, it still does a great job on that part.

All in all, SmokePing has done a great job of being the most famous open source monitoring software. In side it, we can see the flexibility of open source projects. Coming back to our topic, we can see that SmokePing use half of round trip time as the delay, their delay graph might not quite be right. But as opensoure software goes, dedicated network and dedicated devices are understandably not their choice, so do this thesis's solution.

3.2. Time synchronization in the network

Talking about time synchronization in the network, there are interesting problems in a distributed system in which several computers will need to realize the same global time. For example: The make command in Linux systems uses the clock of the machine it runs on to determine which source files that is out of date need to be recompiled; But if the sources reside on a separate file server and the two machines does not have synchronized clocks, the make program might have wrong result.

In a centralized system, the solution is clearly easier. The server will dictate the system time, and some algorithm like Cristian or Berkeley can be used to synchronize the clock. That is the solution for centralized environment, but the solution for a distributed system is not that easy. Some dedicated networks have the luxury of using GPS to receive correct clock on each dedicated device. Other than that, the most popular synchronization solution on the internet is the Network Time Protocol (NTP). This is a protocol based on UDP messages containing time information going over the network. Precision Time Protocol (PTP) is not as famous as NTP, but in local area network, it can achieve clock accuracy in the sub-microsecond range. This makes PTP a very suitable protocol for measurement and control systems.

As the previous section 3.1.2 pointed out, many solutions use the GPS on dedicated devices and in dedicated network to get precise time synchronization. However, with the limit from the customer, it is not easy to get to these two conditions. Therefore, in normal internet or network, NTP is still the better choice. Two particular circumstances does require little to none time synchronization. The first one is that virtual devices are running on the same

server or cloud service, which is pretty much time from one high-processing computer. As a matter of fact, no time synchronization is needed. The second circumstance is that one is running a simulation network on simulation software, which also takes their time from the same source, so time synchronization is also not a problem.

Overall, since we have given some examples of time synchronization in 3.1 when we talk about delay measurement, we will not need more examples on this problem. The two particular points where we do not have to synchronize time may be concentrated on, because our solution are likely going to fall into these points.

4. Proposed solution

In this section, I want to propose a solution for all the challenges mentioned in the above sections. With the spirit in mind that we will not be using dedicated devices or expensive approach, but we can still solve the problem in a satisfied manner.

4.1. Protocol

There are two protocols to be concerned, namely: Sflow and NetFlow. These are the 2 most famous protocols for getting flow information from the network. I will point out the advantages and disadvantages of each protocol, so that we have a better choice on protocol.

First is Sflow. Sflow works on sampling basic, so if one wants to monitor a specific flow, it is a hit and miss behaviour. With our specific task of monitoring only one flow, which is the flow from the sensor to NCS controller, Sflow seems to be not suitable. Moreover, although in some tests I have seen, one can sometimes set Sflow sampling to 1, it's seems to be unrealistic. With Sflow sampling rate set to 1, one is basically mirror all traffic on this interface and send it to the controller. One more disadvantage of Sflow is that the time stamp of Sflow is actually the time that Sflow send the packets to the collector, which gives no valuable information on the actual time of the flow. Going through all the structs of the Sflow datagram, we cannot find the time stamp for when a flow start or end, etc. This information can be added easily into the struct, but unfortunately, the current Sflow version does not have that information yet. All in all, Sflow is not suitable for our task.

NetFlow seems to be more suitable for the solution, not because it's better, but because of NetFlow's mechanism. NetFlow does not sample packets, but monitor all ingress IP packets passing through the interface and aggregate the information of all the flows. Therefore, we will not have the hit or miss behaviour that we have to face with Sflow. NetFlow also offers valuable information in its datagram report. There is the actual time of the flow, although only in millisecond. The header information also gives us the information on the time of the system and the exact time, giving a reference to actually calculate the real time from the SysUptime if necessary. Working around flow and timing information, NetFlow offer rich information for our solution.

In summary, we choose NetFlow as our protocol because of the monitoring mechanism and the information of flow that NetFlow support. And we don't have to worry too much about dedicated devices, because although not being supported by as many vendors as Sflow, NetFlow still have a strong enough list of supporter, like Juniper, Alcatel-Lucent, Huawei, Enterasys, INVEA-TECH, Nortel, VMware, Mikrotik, Open vSwitch, etc. NetFlow can also be installed on PC, both Linux and Windows by the use of probe software. Therefore, we are not having something dedicated here.

4.2. SDN technology

Talking about SDN technology, we cannot avoid mentioning about the rapid evolution going on inside the internet. The old standard infrastructure and standard protocol seems unable to satisfy these evolutions. Look at the booming internet e-shop, market, trade, banking, and the hot virtual money that everyone is talking about, these services need new protocols, new standards, new security, etc. Moreover, the network grows so quick that sooner or later, one has in his hand the task of managing a complex network, with the increasing investment in switch and router, and with the new features, new standards and new protocols, half of which he may never use.

People want a change. First they turn their eye on the hard steel devices of their own, which is not cooperate-able, unfriendly and does not provide a better logic to control the network. There comes the not new idea of dividing the network into two parts: control plane and data plane. It may not be new, but it is the best solution. This separation will add a great flexibility up on the control plane and may allow the network to be programmed to run, to adapt, and to face with different scenario that may happen.

The second problem is the flow, which is the unit of the network. We should have a protocol working between the control plane and the data plane, bringing control information, as well as various statistics for monitoring purpose. Open source, backed up by many big vendors like Cisco, IBM, HP, Alcatel-Lucent, VMware, etc. - OpenFlow quickly become the standard protocol for the communication between southbound interface of the control plane and the data plane.

The third problem is the need of an implementation, or better, a platform that provides modules, interfaces, and even applications. And the big vendors again are providing support

one project, which is OpenDaylight. It has not been standard and is still not fully mature yet, but many companies are starting to use OpenDaylight, especially for OpenStack - an open source cloud computing software platform. As for OpenDaylight controller platform, it provides southbound interfaces and protocol plugins for data plane elements like virtual switches or physical OpenFlow-enabled device interfaces. Therefore, OpenDaylight controller will be our SDN controller, which will provide route control and also the ability to add some modules for NetFlow collector and related modules.

OpenDaylight controller is divided into modules; each has different function, for example: statistics module gives the information of flows, packet loss, number of packets, number of flows, etc. With this flexible, one can add module or use the services from other modules, enable or disable modules, and make restrain on execution of modules. An example of how one module can use other module's service is shown on the figure:

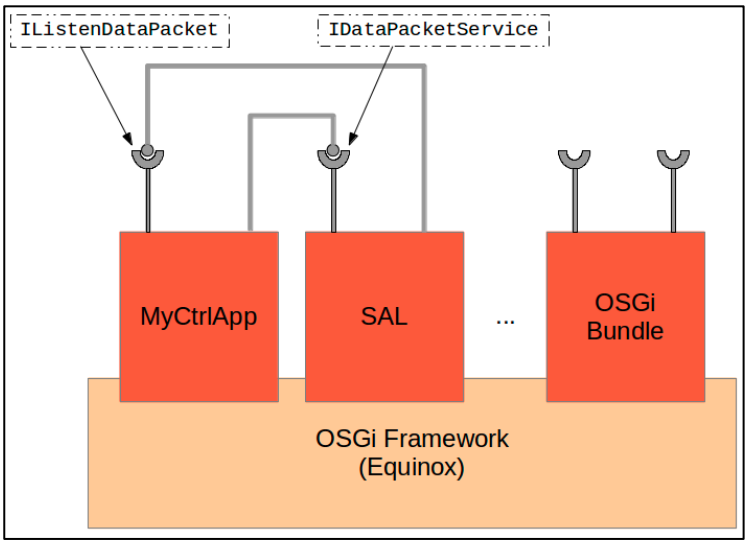

Figure 19: An example of how modules provide services

(13)

These modules and services make OpenDaylight really robust and easy-to-develop. In this solution, we will use OpenDaylight controller and add some modules to the controller for route control, NetFlow data collector, and related modules that will be describe later on.

4.3. Environment

Since we do not have the luxury of having seven OpenFlow enabled switches readily available for testing, we will use one machine with this configuration: Core i7 CPU @ 2.20Ghz x 8; 8GB RAM. In this machine, we will have OpenvSwitch installed, which support OpenFlow and protocols like NetFlow and Sflow. We would have Mininet installed to create a virtual network for the OpenvSwitch to run on. This machine is capable enough to run seven virtual switches created by OpenvSwitch. This machine is installed with Ubuntu version 14.04LTS.s

OpenDaylight controller source code and various OpenDaylight modules will be imported into Eclipse, and this is also the environment we write our modules and run OpenDaylight controller. Eclipse should be configured properly with Maven plugin and many other plugins before the imported OpenDaylight project can work. However, the written modules will be compiled and packaged by Maven outside of Eclipse.

4.4. Delay measurement

In this section, I want to bring out another method of one way delay measurement in which the need for dedicated devices and network is not necessary. This is also the aim of this thesis. This method concentrates on the use of NetFlow information to conduct one way delay measurement.

Although OpenvSwitch supports both NetFlow v5 and IPFIX, I choose NetFlow version 5 because of its popularity. Moreover, IPFIX (the IEEE version of NetFlow v9) has more complex fields and blocks to provide better features like accounting or security. These complexities are not necessary for our operation. Therefore, NetFlow version 5 will be used inside this thesis.

Firstly, let us look at the NetFlow version 5 datagram. This is the header format:

Bytes	Contents	Description
0-1	version	NetFlow export format version number
2-3	count	Number of flows exported in this packet (1-30)
4-7	SysUptime	Current time in milliseconds since the export device booted
8-11	unix_secs	Current count of seconds since 0000 UTC 1970
12-15	unix_nsecs	Residual nanoseconds since 0000 UTC 1970
16-19	flow_sequence	Sequence counter of total flows seen
20	engine_type	Type of flow-switching engine
21	engine_id	Slot number of the flow-switching engine
22-23	sampling_interval	First two bits hold the sampling mode; remaining 14 bits hold value of sampling interval

Table 3: NetFlow version 5 datagram header format

(9)

The datagram header format provides us with two good points. These three points show us the relative information between SysUptime and the current time:

- SysUptime: Current time in milliseconds since the export device booted
- unix_secs: Current count of seconds since 0000 UTC 1970
- unix_nsecs: Residual nanoseconds since 0000 UTC 1970

Following is part of the NetFlow version 5 datagram flow record format:

Bytes	Contents	Description
0-3	srcaddr	Source IP address
4-7	dstaddr	Destination IP address
8-11	nexthop	IP address of next hop router
12-13	input	SNMP index of input interface
14-15	output	SNMP index of output interface
16-19	dPkts	Packets in the flow
20-23	dOctets	Total number of Layer 3 bytes in the packets of the flow
24-27	First	SysUptime at start of flow
28-31	Last	SysUptime at the time the last packet of the flow was received
32-33	srcport	TCP/UDP source port number or equivalent
34-35	dstport	TCP/UDP destination port number or equivalent
36	pad1	Unused (zero) bytes

Table 4: NetFlow version 5 datagram flow record format

(9)

Here we get five more good points:

- dPkts: Packets in the flow
- First: SysUptime at start of flow
- Last: SysUptime at the time the last packet of the flow was received
- srcport: TCP/UDP source port number or equivalent
- dstport: TCP/UDP destination port number or equivalent

Now imagine if we have a tiny portion of the private IP of the network cut out for our usage. And then we use this portion of IP to send out probe messages throughout the network in a slow manner, which will not affect the operation and processing of the devices. These probe messages will be counted as flow and NetFlow will report these messages to us. We check the *srcport* and *dstport* field to verify that this is the package we need. Thus, we have the information about the time of the flow. If we want to measure the delay between two switches, we just probe a message from the controller, going through the two switches and returning to the controller.

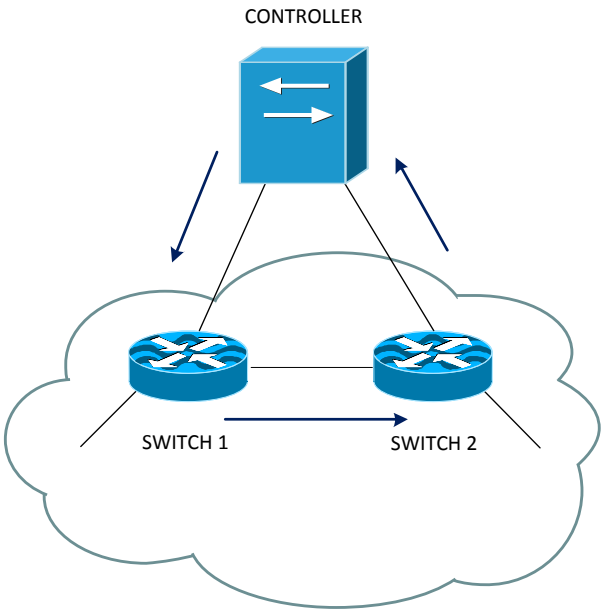

Figure 20: How probe message works

But what we have is the SysUptime, which is not what we really want since they differ from each other as the system start times are not the same.

This is a bit tricky, from the First and Last fields of the flow record format, we have the SysUptime at start and end of the flow. From the flow header format of NetFlow, we have

the relative between SysUptime and unix_secs, which is the current count of seconds since 0000 UTC 1970 and the relative between SysUptime and unix_nsecs, which is the residual nanoseconds since 0000 UTC 1970. Depending on the exact demand, one can choose seconds or nanoseconds to calculate. Here I choose nanoseconds. In the end, we will have the First and Last fields' time converted to milliseconds since 0000 UTC 1970. From this, we can calculate the delay in milliseconds between two switches in one way measurement method.

The equations are as follow:

Equation 2: Difference in seconds

Difference [seconds] = unix_secs [seconds from 0000 UTC 1970] – SysUptime [milliseconds from system start] / 1000

Equation 3: Difference in miliseconds first approach
(This second equation is not really exact since unix_secs is in seconds)

Difference [milliseconds] = unix_secs [seconds from 0000 UTC 1970] * 1000 – SysUptime [milliseconds from system start]

Equation 4: Difference in miliseconds second approach
(A is the next epoch time of 0000 UTC 1970)

Difference [milliseconds] = (A – unix_nsecs) [nanoseconds residual from 0000 UTC 1970] / 1000000 – SysUptime [milliseconds from system start]

Equation 5: Time of flow in miliseconds

Any flow's SysUptime [milliseconds from system start] + Difference [milliseconds] = Time of flow [milliseconds from 0000 UTC 1970]

With these equations, all the time values of the flows have been converted to have the same time reference. And if we want to have the delay in milliseconds, we just have to subtract the two time values.

Equation 6: Delay in miliseconds

Delay [milliseconds] = Time of flow at switch 2 [milliseconds from 0000 UTC 1970] – Time of flow at switch 1 [milliseconds from 0000 UTC 1970]

This is how we solve the one way delay measurement, with probe messages and timing information from NetFlow header and flow record format.

4.5. Time synchronization

As mentioned earlier, the time synchronization problem is not solved completely. Our environment just appears to fall into one of the two particular circumstances which do not require time synchronization. The controller, virtual switches and the workstations are running in the same machine and the time synchronization responsibility is transferred to the machine. Then, all the time on the controller, virtual switches and workstations are synchronized according to the time of the machine.

5. Implementation

In this section, we will look at the implementation inside the SDN controller, which manage the routes inside the network base on the delay information to achieve the minimum load on an optimizing path.

5.1. Module function

Following is the function of implemented modules inside the SDN controller.

- Map Optimizer: The map optimizer module is the module where the network is optimized in the first place. Since we only need to feed packets to paths which connect the NCS controller and the sensors, other paths are not being considered. Furthermore, we can also implement other optimization before sending the optimized map to other modules. Figure 21 denotes the full network links. Note that the virtual SDN controller is connected to every switches inside the network and is not depicted here.

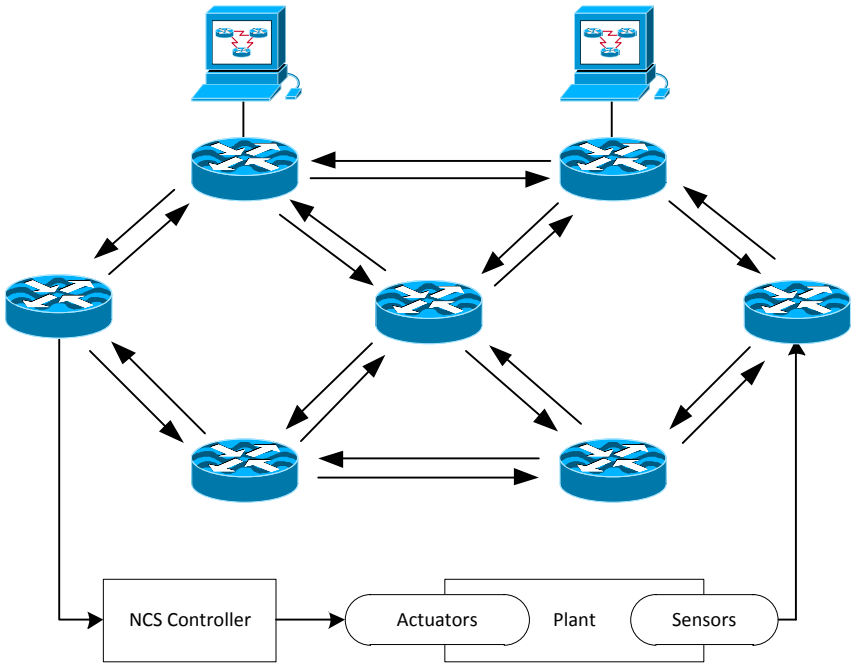

Figure 21: Full network links

Figure 22 denotes how the considered links are taken out to form an optimized map.

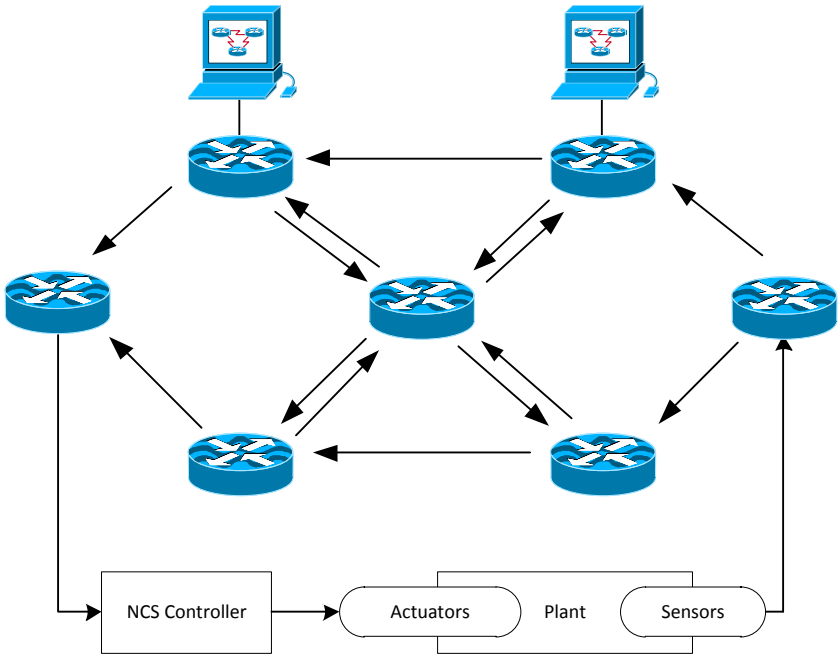

Figure 22: Network links after optimization

Figure 23 denotes an example of further optimization inside the network.

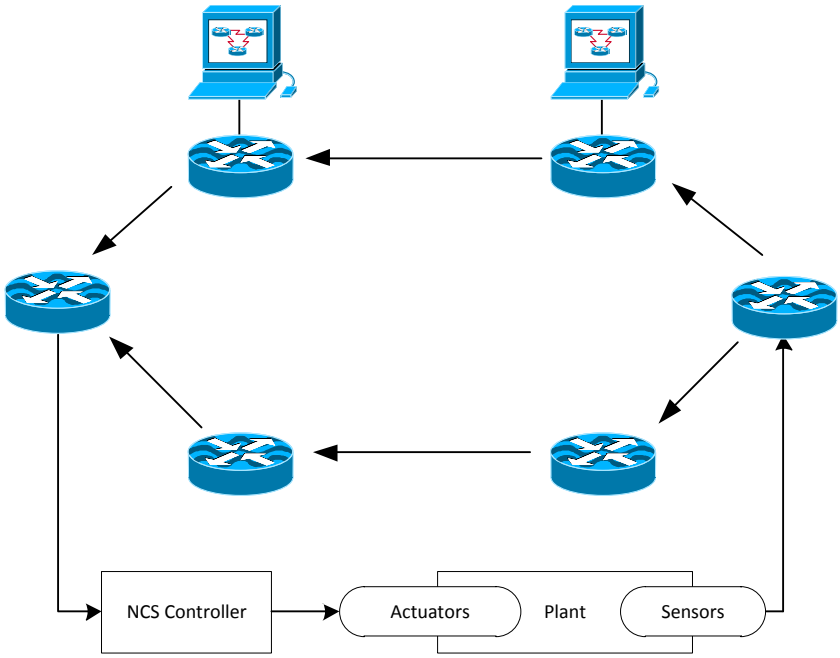

Figure 23: Further optimization with only two switches in between

- Route Optimizer: The optimizing algorithm is done inside this module. It is another implementation just like the one we see in (1). This module takes in a special map from the NetFlow Collector, calculates the best route and deploys the result. If no route is in accordance, the module will not change the current route. If the module does not receive any delay information from the NetFlow Collector, it will apply a temporary shortest route.

- NetFlow Collector: This module has to collect NetFlow messages from the switches and update the delay information into a map called NCSMap. This map will then be transferred to the Route Optimizer when asked.

- Flow Feeder: The Flow Feeder is responsible for feeding the messages into the network according to the map provided by the Map Optimizer.

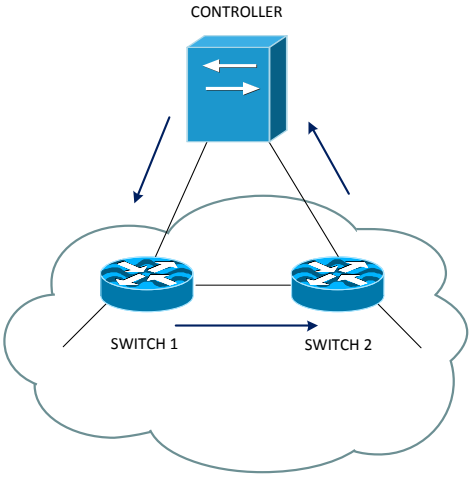

Figure 24: Flow feeder feeding messages to measure delay between two switches

5.2. Interaction

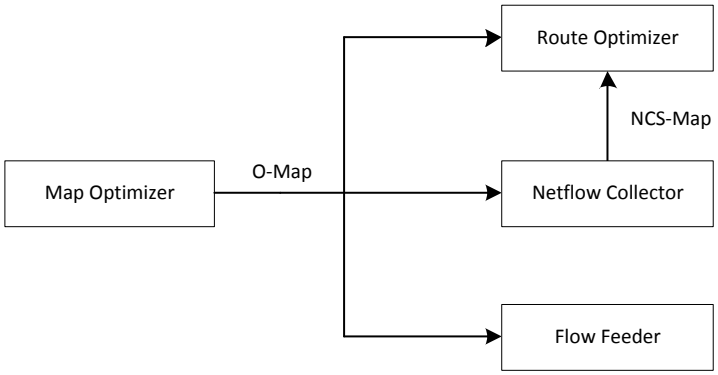

Figure 25: Module interaction

After the network congested, the Map Optimizer will receive the topology information from the service in SDN controller. It will create an NCSMap with necessary links. Further optimization can be done here. The Flow Feeder will then gradually feed messages to the network according to the map. This feeding of messages are repeated to provide sufficient delay information. The switches will periodically send flow summary to the NetFlow Collector. The collector will then receive and calculate the delay information into the NCSMap. The Route Optimizer get the NCSMap periodically from NetFlow Collector and optimize the route according to the algorithm provided.

5.3. Necessary services

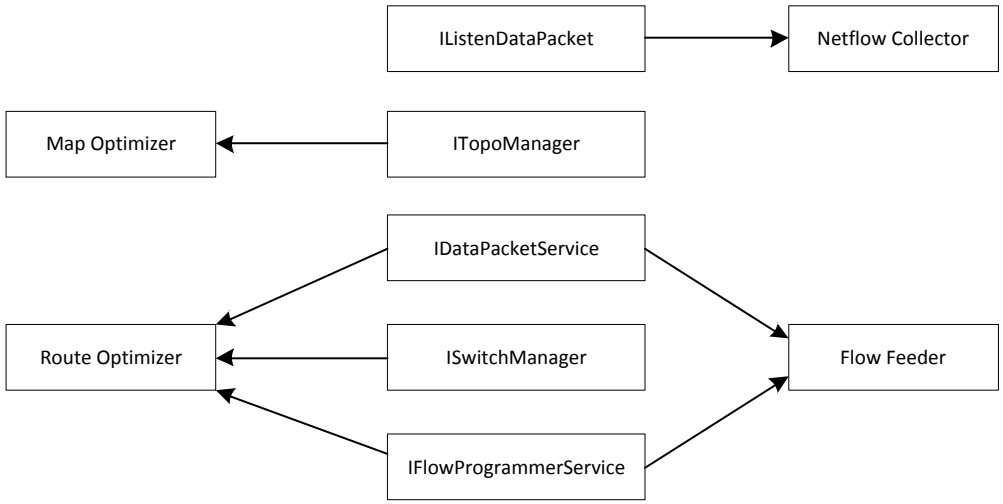

Figure 26: Necessary services

Following are the most important services necessary for each module:

- The Map Optimizer module will need to optimize the map base on a topology obtained from the ITopoManager service.
- The Route Optimizer module will need to have ISwitchManager to get some topology information of layer 3 switches. The IFlowProgrammerService is used to send flow customization to the switches inside the network. In order to send these information to the switches, it need the encoding from IDataPacketService.
- The NetFlow Collector module need to listen to packets coming to the controller, hence IListenDataPacket is required.
- The Flow Feeder module have to transfer packets on the network so it needs the encoding from IDataPacketService. It also needs IFlowProgrammerService to send flow customization to the switches.

6. Summary

The proposed solution provides another approach for the optimization of routing using OpenFlow. It utilized the promising OpenDaylight controller to implement the algorithm. It also provide a new way to measure one-way delay between two switches. However, this delay is still affected by the buffer delay and is not exact. Exact one-way delay measurement still require a dedicated device with dedicated hardware at the buffer to determine the timing. Moreover, the time synchronization is also an unsolved problem. The optimization algorithm will need further research since it only fits in small network. With big network, the calculation requires a lot of time. Therefore, we still need further study inside these topics.

Reference list

1. *A Network Abstraction for Control Systems.* **Ben W. Carabelli, Frank Dürr, Boris Koldehofe, Kurt Rothermel.** 2014.

2. *One-way Delay Measurement and Characterization.* **Ana Hernandez, Eduardo Magaña.** 2007.

3. *One-Way Delay Measurement Techniques.* **Accedian Networks.** Jan 2012.

4. *Monitoring latency with OpenFlow.* **Bouet, Kevin Phemius and Mathieu.** 2013. ISBN 978-3-901882-53-1.

5. *Measuring Delay, Jitter, and Packet Loss with Cisco IOS SAA and RTTMON.* **Cisco Systems, Inc.** Oct 25, 2005.

6. **Peter Phaal.** sFlow blog. [Online] blog.sflow.com.

7. **InMon Corporation.** sFlow. [Online] sFlow.org.

8. **SolarWinds.** NetFlow Traffic Analyzer. [Online] go.solarwinds.com.

9. *NetFlow FlowCollector Installation and User's Guide.* **Cisco Systems, Inc.**

10. *Measuring Latency Using NetFlow.* **Luca Deri, Michael Patterson.** s.l. : Plixer International.

11. **ntop project.** nProbe™ v7. [Online] ntop.org.

12. **Tobias Oetiker, Niko Tyni, David Schweikert, et.al.** SmokePing. *OETIKER+PARTNER.* [Online] oss.oetiker.ch/smokeping/.

13. **Frank Dürr.** Networked and Mobile Systems. [Online] frank-durr.de.