Estimating the Optimal Model for Layer 4 Switching in IPv6 and IPv4

Daniel ZINCA, Virgil DOBROTA, Cristian Mihai VANCEA
Technical University of Cluj-Napoca, Department of Communications, 26 Baritiu Street, 3400 Cluj-Napoca, Romania,
Tel/Fax: +40-64-197083, E-mail: {Daniel.Zinca, Virgil.Dobrota, Mihai.Vancea}@com.utcluj.ro

1. Introduction and Layer 4 switching concepts

This paper is focused on a Layer 4 switching experiments of IPv6 over Fast Ethernet, running under Windows 2000 Professional. After our studies on IPv4 over ATM with TCP relaying [3], firstly presented at IEEE LANMAN'99, we are continuing to evaluate the performances at the interface between the applications and the non-blocking stream-oriented sockets in TCP/IP. The first major objective is to get consistent arguments for the IPv6 versus IPv4 debate, even that the implementation phase of the new version of the Internet Protocol is under progress. Our approaches are different from those discussed in [1], as we don't propose any changes in TCP implementations. Being aware of basic congestion control issues (such as slow start, congestion avoidance, fast retransmit and fast recovery) outlined in [2], we are evaluating an intelligent access to Internet, based on a feedback from the network. We propose a method for measuring the RTT *(Round Trip Time)* at the Application Layer, with the accuracy provided by the CPU's clock period (i.e. 1 nanosecond for 1GHz), relying on Intel's Pentium RDTSC *(Read Time Stamp Counter)* instruction. Obviously this performance is requested either by new emerging technologies, such as IP over Gigabit Ethernet or IP over DWDM, either by IP QoS features, in general.

Due to the fact that the modeling of the self-similar behavior of a given network is rather difficult to be performed (evenly on-line or off-line), the idea is to involve departure scheduling mechanisms for APDUs *(Application Protocol Data Units)* at the originating client and, eventually, a Layer 4 switching scheduling at the server site. The real-time source model is described by the general parameters such as: a (idle-to-active changing rate), b (active-to-idle changing rate) and D (transfer bit rate). Depending on the application type, there are specific variables such as: number of bursts (for ON/OFF sources), peak-bit-rate (for video sources) etc. [4]. Based on the feedback from the real network, a comparison between the estimated RTT and the measured RTT is performed, the model being dynamically adjusted by changing the input parameters (for instance the actual transfer bit rate). A core node in the network, acting as a server, switches the TCP segments from an incoming socket to an outgoing socket. The edge nodes, acting as clients, exchange both data and signaling through the previously defined Layer 4 switch. All the experiments, concerning both IPv6 and IPv4 presented herein, involved Fast Ethernet technology and Microsoft's Windows 2000 Professional. An updated version of the software tool from [3] offered the facilities to evaluate the sending time, the receiving time and the elapsed time at the interface between the application and the non-blocking stream-oriented socket in TCP/IP. Obviously each Layer 3 protocol version requested its own TCP implementation on top of it. Recently, Cisco Systems Inc. has introduced its own concept of Layer 4 switching, rather different from that one we are using in this paper. Cisco's Layer 4 switch is a re-distributor of the requests and hits from clients evenly among all the server in the server farm, in order to achieve a balanced load for each server [9].

2. Measuring the RTT at the Application Layer: protocol description

The newly proposed 25…1024-byte APDU protocol for measuring the RTT at Layer 7 does not involve any changes of the existing protocols in Internet. It is realized by software at the interface between Layer 4 and the Application Layer and is carried out on a "signaling socket" (different from "data socket"). The method is based on a bi-directional exchange of APDUs between the Client 1 (source) and the Client 2 (destination), through the Server (Layer 4 switch), with the following significance of the fields (see *Slide 3*): *Type* is similar to the field *Stratum* used by NTP; *Sending Timestamp* is ts_{12_i} from Client 1 to Client2, ts_{21_i} from Client 2 to Client 1; *Sending Sequence Number (SSN)* and *Receiving Sequence Number (RSN)* are requested by UDP-based applications only; *Receiving Timestamp* is ts_{12_f} or ts_{21_f}; *Sending Processor's Frequency* is given in MHz, f_{CPU1} for the Client 1, f_{CPU2} for the Client 2; PAD (0 or 7992 bits) represents additional 7992 bits of 00h for Ethernet frame-based technologies, because the 1024-byte APDU gives a more accurate evaluation of the overall throughput. The 25-byte APDU (0 bits of PAD) is recommended for technologies such as ATM. At the server site, the switching time is defined as the interval since the reception of the first TPDU started until the transmission of the last TPDU ended. Note that there is no synchronization process involved in our approach, as we are using the timestamps for fine measurements purposes only. Furthermore, the timestamps are not added, as usually, to the user data segments, being encapsulated and sent through a distinct socket. The client processes of interpreting the feedback from the real network are running on the edge nodes, at the Application Layer, whilst the server ones perform a Layer 4 switching in a core node, but without additional congestion control algorithms. All of these do not exclude the involvement of the extensions to the standard TCP, such as window scaling and timestamps for window sizes of more than 64KB, as in [2], or selective and delayed acknowledgments, as in [1]. We rely on these new implemented features, provided also by the current TCP implementation within Windows 2000 Professional, and we paid a special attention to avoid the reducing of performances due to Nagle algorithm. Note also that our approach is different from NETBLT protocol, which is a transport level protocol designed for a bulk data transfer, but involving two strategies for flow control: one internal, based on a burst rate, and one at the client level, based on a burst size [6]. Other differences from NETBLT are related to the fact that no negotiations are performed between the sender and the receiver during the transmission, whilst the flow control is provided by the Layer 4 implementation, based on the indirect TCP connections through the switch.

3. Experimental results and conclusions

A consistent set of models (starting from 1 Mbps up to 100 Mbps transfer bit rate) were employed, considering $a = 0.001$ and $b = 0.005$. Following the same methodology, as for IP over ATM in [3], the result was a total disappointment: the TCP entities, for both IPv6 and IPv4 at 100 Mbps, were able to follow almost none of the models (even at 1 Mbps !). The first optimization was to split the bursts into fragments not greater than 8192 bytes, within the same period ON+OFF established by the model. This time was much better: for IPv4, the sending TCP entity was able to follow all the models (including 100 Mbps), at a CPU's frequency of at least 366 MHz. Due to the physical limitations of the network, according to the switching times of PDUs, measured at the server site, and the receiving times at the destination client, the upper bound was about 38 Mbps, whilst the higher transfer rate for IPv6 was 36 Mbps (that is about 31 Mbps average throughput for the given a and b). In order to avoid these measurements in practice, we proposed an algorithm for calculating the RTT at Layer 7, to anticipate the previously mentioned results and to validate the conclusions. The idea was that the application will obtain a consistent view of the current client-server-client link state. It will be able to estimate if the real-time stream could be delivered by the existing network in due time, without involving resource reservation protocols or other QoS facilities. On the other hand, it is for further work to involve scheduling at the server site, too. The APDU for RTT was encapsulated in a TCP segment, then in an IPv6/IPv4 datagram, then in a Fast-Ethernet frame. According to [4],[7], we had to evaluate the following (see *Slide 4*): access delay (the time requested to access the socket); packetization delay (the time needed to encapsulate the socket's data into a Layer 2 frame); transmission delay (the time needed to send the frame between Layer 2 and Layer 1, 13.6 m s in IPv4, respectively 16.8 m s in IPv6, at 100 Mbps); propagation delay (depending on the distance and on the media); reassembly delay; queueing delay and Layer 4 switching delay. Concerning the relevance of the measurement on the signaling socket for the real-time transmission on the data socket, the experiments proved that an APDU for RTT should not exceed 1024 bytes (in order to be transmitted in one frame), whilst for data is better to have fragments not greater than 8192 bytes, whenever it's possible. The compromise is based on the observation that the major differences between the exchanges on the signaling and on the data socket are related to the transmission delays mainly. We can extrapolate the results obtain for the 1024-byte RTT to establish an upper bound for the 8192-byte APDU. As a lot of experiments are still under progress, the only one preliminary remark concerning the overall throughput is that the current implementation of IPv6 for Windows 2000 Professional has no advantages, comparing to IPv4. On the other hand we managed to perform a 8192-byte Layer 4 switching in 59 m s for IPv6, instead of 111 m s for IPv4. Unfortunately, for smaller PDUs, it seems that better performances are provided by IPv4.

References

[1] M. Allman, A. Falk, "On Effective Evaluation of TCP", *Computer Communication Review, ACM-SIGCOMM*, Volume 29, Number 5, October 1999, pp. 59-70

[2] M. Allman, "A Web Server's View of the Transport Layer", *Computer Communication Review, ACM-SIGCOMM*, Volume 30, Number 5, October 2000, pp. 10-20

[3] V. Dobrota, D.Zinca, C.M. Vancea, A. Vlaicu, "Layer 4 Switching Experiments for Burst Traffic and Video Sources in ATM", *Digest of the 10th IEEE Workshop on LANMAN'99*, Sydney, 21-24 November 1999, pp.66-69.

[4] V. Dobrota, *Retele digitale in telecomunicatii. Volumul 2: B-ISDN cu ATM, Sistemul de semnalizare cu canal comun SS7*. Editura Mediamira, Cluj-Napoca 1998.

[5] D.L. Mills, "Network Time Protocol (V.3) Specification, Implementation and Analysis ", *RFC 1305*, March 1992.

[6] D.D. Clark, M.L. Lambert, L. Zhang, "NETBLT: A Bulk Data Transfer Protocol", *RFC 998*, March 1987

[7] F. Tobagi, I. Dalgic, "Performance Evaluation of 10Base-T and 100Base-T Ethernets Carrying Multimedia Traffic", *IEEE Journal on Selected Areas in Communications*, Vol.14, No.7, September 1996, pp. 1436-1454.

[8] ***, *http://msdn.microsoft.com/downloads/sdks/platform/tcpip6.asp*

[9] ***, *http://www.cisco.com/univercd/cc/td/doc/product/l4sw/index.htm*

Estimating the Optimal Model for Layer 4 Switching in IPv6 and IPv4

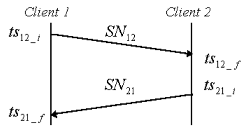
D. Zinca, V. Dobrota, C. M. Vancea
Technical University of Cluj-Napoca
26 Baritiu Street, 3400 Cluj-Napoca, Romania
E-mail: Daniel.Zinca@com.utcluj.ro
http://www.utcluj.ro/utcn/eltc

SCHEME OF TALK

- Conversion from IPv4 to IPv6
- Layer 4 switching concepts
- Application Protocol Data Unit RTT
- Related work:
  - **Extensions of standard TCP**
  - **Synchronization protocols**
  - **NETBLT**
- Experimental results: IPv6 versus IPv4
- Conclusions

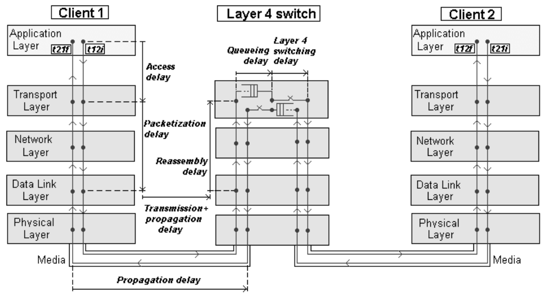
APDU (Application Protocol Data Unit) RTT (1)

Byte 0	*Bytes 1 ... 8*	*Bytes 9 and 10*	*Bytes 11 ... 18*	*Bytes 19 and 20*	*Bytes 21 ... 24*	*(Bytes 25 ... 1023)*
Type	**Sending Timestamp**	**Sending Sequence Number**	**Receiving Timestamp**	**Receiving Sequence Number**	**Sending Processor's Frequency**	**PAD (optional)**
(8 bits)	**(64 bits)**	**(16 bits)**	**(64 bits)**	**(16 bits)**	**(32 bits)**	**(0 or 7992 bits)**

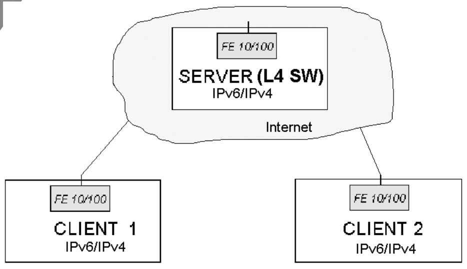


$$RTT = \frac{t_{21\_f} - t_{12\_i}}{f_{CPU1}} - \frac{t_{21\_i} - t_{12\_f}}{f_{CPU2}}$$

APDU RTT (2)



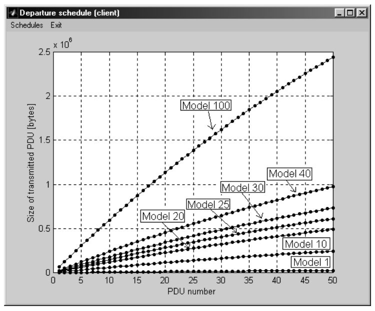
Testing Configuration

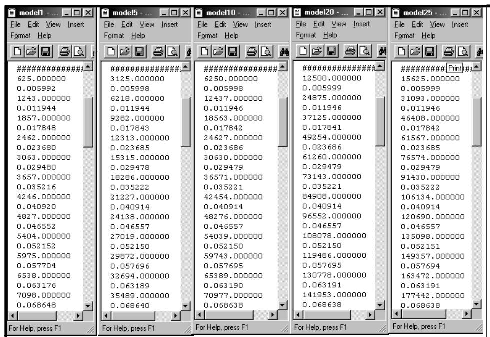


Departure schedules (1)

$\alpha = 0.001$
$\beta = 0.005$
50 bursts

Model 10 =>
$\dot{D}$ =10 Mbps





```
model1
################
625.000000
0.005992
1243.000000
0.011944
1857.000000
0.017848
2462.000000
0.023680
3063.000000
0.029480
3657.000000
0.035216
4246.000000
0.040920
4827.000000
0.046552
5404.000000
0.052152
5975.000000
0.057704
6538.000000
0.063176
7098.000000
0.068648
```

```
model5
################
3125.000000
0.005998
6218.000000
0.011944
9282.000000
0.017843
12313.000000
0.023685
15315.000000
0.029478
18286.000000
0.035222
21227.000000
0.040914
24138.000000
0.046557
27019.000000
0.052150
29872.000000
0.057696
32694.000000
0.063189
35489.000000
0.068640
```

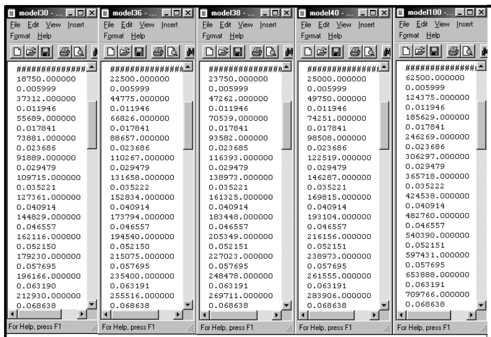
```
model10
################
6250.000000
0.005998
12437.000000
0.011946
18563.000000
0.017842
24627.000000
0.023686
30630.000000
0.029479
36571.000000
0.035221
42454.000000
0.040914
48276.000000
0.046557
54039.000000
0.052150
59743.000000
0.057695
65389.000000
0.063190
70977.000000
0.068638
```

```
model20
################
12500.000000
0.005999
24875.000000
0.011946
37125.000000
0.017841
49254.000000
0.023686
61260.000000
0.029479
73143.000000
0.035221
84908.000000
0.040914
96552.000000
0.046557
108078.000000
0.052150
119486.000000
0.057695
130778.000000
0.063191
141953.000000
0.068638
```

```
model25
#########Print##
15625.000000
0.005999
31093.000000
0.011946
46408.000000
0.017842
61567.000000
0.023685
76574.000000
0.029479
91430.000000
0.035222
106134.000000
0.040914
120690.000000
0.046557
135098.000000
0.052151
149357.000000
0.057694
163472.000000
0.063191
177442.000000
0.068638
```

Departure schedules (2)

The 1st entity is the number of bytes during the 1st burst (ex: 625 B) , the 2nd entity is the duration of the 1st period ON+OFF (ex: 0.005992 s.) etc.



```
model30
################
18750.000000
0.005999
37312.000000
0.011946
55689.000000
0.017841
73881.000000
0.023686
91889.000000
0.029479
109715.000000
0.035221
127361.000000
0.040914
144829.000000
0.046557
162116.000000
0.052150
179230.000000
0.057695
196166.000000
0.063190
212930.000000
0.068638
```

```
model36
################
22500.000000
0.005999
44775.000000
0.011946
66826.000000
0.017841
88657.000000
0.023686
110267.000000
0.029479
131658.000000
0.035222
152834.000000
0.040914
173794.000000
0.046557
194540.000000
0.052150
215075.000000
0.057695
235400.000000
0.063191
255516.000000
0.068638
```

```
model38
################
23750.000000
0.005999
47262.000000
0.011946
70539.000000
0.017841
93582.000000
0.023685
116393.000000
0.029479
138973.000000
0.035221
161325.000000
0.040914
183448.000000
0.046557
205349.000000
0.052151
227023.000000
0.057695
248478.000000
0.063191
269711.000000
0.068638
```

```
model40
################
25000.000000
0.005999
49750.000000
0.011946
74251.000000
0.017841
98508.000000
0.023686
122519.000000
0.029479
146287.000000
0.035221
169815.000000
0.040914
193104.000000
0.046557
216156.000000
0.052151
238973.000000
0.057695
261555.000000
0.063191
283906.000000
0.068638
```

```
model100
################
62500.000000
0.005999
124375.000000
0.011946
185629.000000
0.017841
246269.000000
0.023686
306297.000000
0.029479
365718.000000
0.035222
424538.000000
0.040914
482760.000000
0.046557
540390.000000
0.052151
597431.000000
0.057695
653888.000000
0.063191
709766.000000
0.068638
```

Departure schedules (3)

The 1st entity is the number of bytes during the 1st burst (ex: 18750 B) , the 2nd entity is the duration of the 1st period ON+OFF (ex: 0.005999 s.) etc.

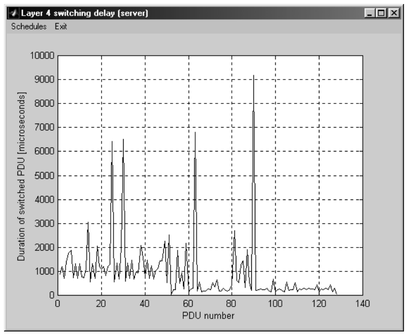
EXPERIMENTAL RESULTS (1)

Application's buffer size [Bytes]	Average switching time (IPv6) [μ s]	Average switching time (IPv4) [μ s]	Average switching speed (IPv6) [Mbps]	Average switching speed (IPv4) [Mbps]	Average receiving time (IPv6) [μ s]	Average receiving time (IPv4) [μ s]
8192	59	111	1083.38	575.84	102	217
5000	317	266	201.64	240.30	565	721
3000	486	333	131.52	191.95	1371	1074
1500	N.A.	617	N.A.	103.59	N.A.	2458
750	1249	937	51.17	68.21	5498	4945
375	2601	1738	24.57	36.77	10843	9502

Testfile1 (7990 bytes), Client 1 -> server -> Client 1, without model. The average sending time/ throughput: 264 μs/242.12 Mbps for IPv6 and 165 μs/387.39 Mbps for IPv4 (CPU> 366 MHz)

EXPERIMENTAL RESULTS (2)

> Layer 4 switching delay
> Testfile2 (240,118 bytes)
> Client 1 -> server -> Client 2
> Model 30
> IPv6/100 Mbps
> Throughput = 26.52 Mbps



Estimation of the optimal model for IPv6 at 100 Mbps

- **RTT = *2159.077 μs* (between Client 1 and Client 2)**
- **4 x Transmission time for 1024-byte APDU encapsulated in 1104 bytes = 4 x 88.32 = *353.28 μs***
- **Transmission time for the 8192-byte APDU encapsulated in 8 frames of 1104 bytes = 8 x 88.32 = *706.56 μs***
- **The <u>estimated maximum transfer bit rate</u> is < (8192 x 8) / [(2159.077-353.28)/2+706.56] = *40.71 Mbps***
  i.e. Model 40 is the upper bound !!!

***Note:* Experimentally, for the given =0.001, ß=0.005, the highest model followed by the TCP entity was Model 36.**

CONCLUSIONS

> The optimal model for Fast Ethernet was estimated based on:
- ***<u>1024-byte RTT</u>* => gives the highest transfer bit rate**
- **bursts in *<u>fragments < 8192 bytes</u>* => capabilities to follow the models**
- ***<u>application's buffer = 8192 bytes</u>* => better results for Layer 4 switching in IPv6**

> Further work:
- **other TCP implementations, competing traffic flows**
- **idle-to-active, active-to-idle changing rates variables**
- **optimum period for RTT updating**
- **scheduling for Layer 4 switching**