TKN Telecommunication Networks Group

Technical University Berlin

Telecommunication Networks Group

Adaption of a webtraffic generator to a WDM ring network

Sven Wiethoelter, Hagen Woesner

{wiethoel, woesner}@ft.ee.tu-berlin.de

Berlin, February 2003

TKN Technical Report TKN-03-13

TKN Technical Reports Series

Editor: Prof. Dr.-Ing. Adam Wolisz

Contents

Chapter 1

Large-Scale-Webtraffic Generator

1.1 Model

For the analysis of the WDM network, a load model was needed to consider the MAC-fairness algorithm as well as the local fairness. Today's backbone traffic is assumed to be self-similar. [4] describes that "self-similar traffic can be produced by multiplexing ON/OFF sources that have a fixed rate in the ON periods and ON/OFF period lengths that are heavy-tailed." A large number of ON/OFF sources is needed to obtain sufficient throughput. These ON/OFF sources are represented by a client/server model with a large number of clients and servers. The behavior of the ON/OFF sources is Poisson modelled. To achieve self-similarity, the size of requestable files is modelled with the Pareto distribution.

1.2 Description of the large-scale webtraffic generator

We use the network simulator ns2.1b8a [1] for our simulations . It is a discrete event simulator, which has been developed at the University of Southern California. Part of the network simulator is the large-scale-webtraffic generator. This generator provides self-similar traffic with multiple ON/OFF sources. Therefore we decided to adopt this generator to the WDM-network.

Basically the generator creates multiple client-server connections. This means that a client sets up a TCP-connection and launches a single request to a server. After receiving the request the server sends the requested webfile to the client. The behaviour of the server in relation to the processing time of a request is not modelled in this traffic generator. Here a server answers without any processing time after receiving a request.

In this implementation a webfile is represented by an object. One or more objects represent a page. For every object a client sets up a new TCP connection. In the case of more than one object per page HTTP1.0 would be modelled. HTTP1.1 would be simulated with only one object per page. A defined number of pages is grouped to one session. A session itself is assigned to one client, which queries all pages and objects of this session from different servers. Therefore it makes sense to choose a greater number of sessions than clients.

The subdivision into sessions, pages and objects is important for controlling generator's behaviour. Its parameters are listed below:

- number of sessions
- inter-session time
- number of webpages per session
- inter-page time
- number of objects per page
- inter-object time
- object size
- shape

The number of sessions, pages and objects are fixed parameters. The inter-session time, the inter-page time as well as the inter-object time defines the time between the starting points of sessions, pages or objects. They are average values of the Exponential Distribution. If these parameters are chosen very small, sessions, pages and/or objects will be worked off parallel. To obtain the behavior of real web-traffic the object size is ParetoII-distributed. The parameter "object size" is an average value of this distribution. The shape (α) of ParetoII is fixed to 1.2. Please visit [2] for further informations about ParetoII and its choosen parameters.

Each TCP-connection is modelled with two Reno-TCPs. Since the generator was developed in a time where ns could simulate only one-way TCPs, two Reno-TCPs were the only solution for achieving Full-TCP. The ns's Reno-TCPs work only with two packet sizes: 40Byte-packets for connection establishment and acknowledges and a fixed size for data packets. Because of a lot LAN-size packets in the Internet we have chosen a packet size of 1.5kB.

An example script of the traffic generator can be found in the directory `../nsx.x/tcl/ex/`. The file `varybell.tcl` contains settings of the topology. The parameters of the generator and the example simulation are included in `large-scale-webtraffic.tcl`. The most important functions are implemented in `webtraf.cc` and `webtraf.h`. These functions interact with the Tcl-functions in `../nsx.x/tcl/webcache/webtraffic.tcl`. They are responseable for TCP connection establishment and release as well as for launching and replying of requests. `webtraffic.tcl` itself accesses on standard-ns C++-functions. Tcl/Tk has one important disadvantage: it is not able to free objects at runtime. Because of the alternation between Tcl/Tk and C++-interfaces for every sent packet, memory usage increases by simulating with the generator. To avoid a fast growing memory usage at runtime the parameters must be chosen carefully!

Chapter 2

Generator and WDM-Ring Network

For analyzing a WDM-ring network with a maximum throughput of 622Mbit/s under heavy TCP load an adaption of the large-scale-web-traffic generator was necessary because the default-adjusted generator reached only a throughput of 1Mbit/s.

2.1 Parameters of the adopted Generator

- number of sessions: 1000 (default 400)
- inter-session time: 0.0001s (default: 1s)
- number of webpages per session: 250 (default)
- inter-page time: 1s (default: 15s)
- number of objects per page: 1 (default)
- inter-object time: 0.01s (default)
- object size: 40 packets (default: 12 packets)
- shape: 1.2 (default)

Since HTTP1.0 is a part of history the number of objects is set to 1. For reaching a throughput of 600Mbit/s a high number of sessions, clients and servers is needful. For minimizing the dynamic behavior of the network at the beginning of the simulation a very small inter-session time is required. Practically all sessions are started at once. We have decreased the inter-page time for reaching higher throughputs. To keep a useful exponential distributed "think time" of the clients we have not choosen a value smaller than 1. Because of only one object per page the inter-object time has no effect on the simulation.

2.2 Simulation with one hop in the WDM network from servers to clients

2.2.1 Topology

For testing the generator as well as for analyzing the WDM network with TCP traffic we put the servers and the clients on different sides of the network (see Fig. 2.1). The clients are connected via their parent-node 5 to the network. In Fig. 1 the clients are abbreviated as "C". Overall there are 600 clients connected to node 5.

The servers correspond with the network via parent-node 6 and the intermediate nodes 7-16. 10 servers are contacted to each intermediate node. This leads to 100 servers overall. The subdivision in parent-nodes and intermediate nodes is necessary to prevent a bottleneck before the packets arrive at the WDM network.

The nodes 0 and 2 to which the parent-nodes of the clients respectively the servers are connected will be appointed below as client-node resp. server-node.

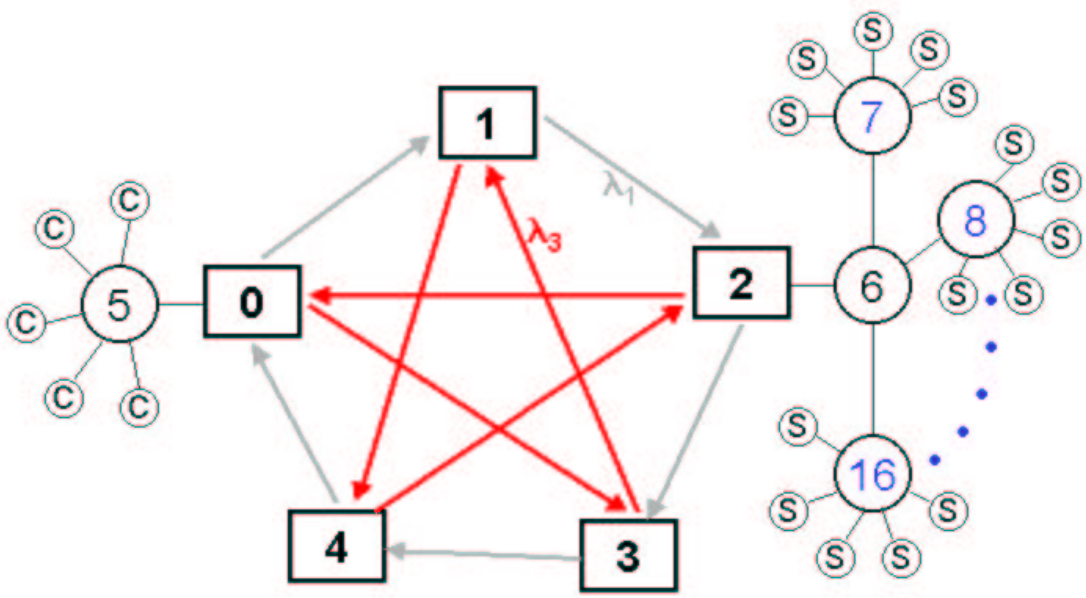

Figure 2.1: WDM-Ring Network with added clients and servers of the generator

At first we simulated the WDM Rings with the wavelengths 1 and 3. That equals the topology of unidirectional ring and star. This is the easiest observation because the responses of the servers have only one hop to complete on the star for reaching the client-node.

2.2.2 Link Lengths and Delays, Throughputs and TCP-Window

For a maximum throughput an optimum TCP-Window is required. Therefore we have determined the bandwidth-delay product for each link inside resp. outside of the WDM network.

Beneath we have determined the link length L of each segment with L=delay*$\frac{2}{3}$*c_0 (c_0 being the light velocity in vacuum). Between servers and clients (outside of the WDM network):

- server → intermediate node: L=3km, 15μs*400Mbit/s = 6kbit
- intermediate → parent node: L=10km, 50μs * 600Mbit/s = 30kbit
- parent node → server node: L=10km, 50μs * 1.2Gbit/s = 60kbit
- client node → parent node: L=10km, 50μs * 800Mbit/s = 40kbit
- parent node → client: L=4km, 20μs * 400Mbit/s = 8kbit

In the WDM network the bandwidth-delay product depends on the number of hops. For one hop it is Link + Delay-Line: (200μs+$\frac{9216*8}{622Mbit/s}$)*622Mbit/s≈318.53μs*622Mbit/s=198.126kbit Both directions result in 3 hops on the rings: 3*198.126kbit+288kbit=882.378kbit. Therefore a TCP-Window of approximately 74 packets (each containing 1.5kB) is required. To assure that the windows is big enough for other ring topologies too we chose a windows of 100 packets.

2.2.3 Simulation Goals

The aims of this simulation:

- finding an appropriate number of sessions for achieving a high throughput on the ring (≈600Mbit/s) with a payload up to 50%
- minimizing the dynamic behavior of throughput and delay for early termination (because otherwise the simulation results in high memory usage).

2.2.4 Simulation Structure

For our stochastic analysis, we use Akoara [3], [5] which has been developed at the University of Canterbury in Christchurch, New Zealand. Akoroa speeds up stochastic simulations and provides MRIP (Multiple Replications In Parallel). The use of Akaroa avoids the analysis of large trace files due to "real-time simulation data output analysis" and "automatic run length control" [3].

We have chosen a confidence level of 95% and a precision of 5% for our observed parameters throughput and delay of the Macs 0 and 5 (that equals the sending MAC-entities in client node 0 and server node 2). These parameters are observed by Akaroa. It terminates the simulation when the confidence level for both parameters is achieved.

Because of the fixed length of the Jumbo packets (9216 bytes) in the ring network aggregating is required. To distinguish the payload from the brutto-load trace files of the link 2→0 are dumped (can be found on the CD in the directory `/traces/mylink`). In these files the 1.5kByte "small" data packets in the queue of the Mac as well as the later built Jumbo packets are itemized.

To receive the throughput of each server response (each TCP-connection) the verbose mode in the file `../nsx.x/tcl/webcache/webtraffic.tcl` has to be turned on. Then each done-response is listed in a trace file (can be found on the CD in the directory `/traces/done-resp/`).

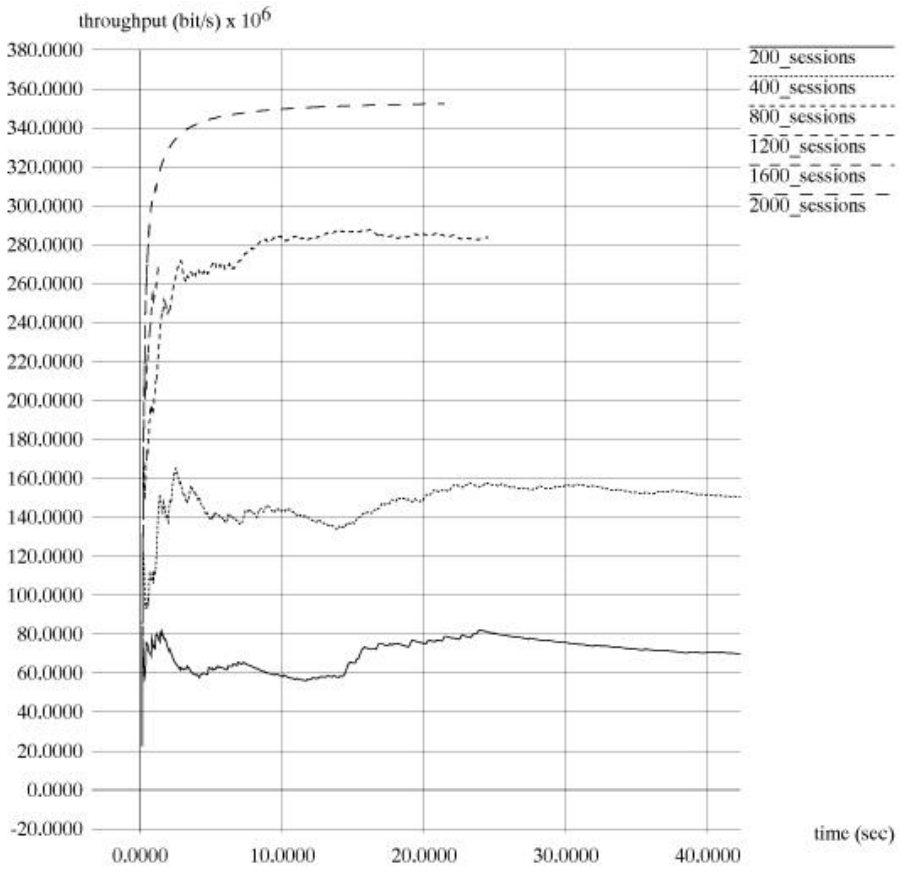

Figure 2.2: throughput of simulations with 200, 400, 800, 1200, 1600 and 2000 sessions

2.2.5 Carrying Out and Results

To determine the optimum number of sessions for a high a throughput we ran the simulation with a number of sessions from 100 to 2000 with a step size of 100. The results of the simulations with 200, 400, 800, 1200, 1600 and 2000 sessions are shown above in figure 2.2 by plotting the throughput over the simulated time. The x-scale has been fixed thereby around 40 seconds. In figure 2.2 we have left out the simulations with other session-numbers to get a general idea of system's behavior. Data for this figure has been taken out of the "mylink" traces by counting the number of sent 1.5kB packets/time. It is conspicuous that an increasing number of sessions deliver not only a higher throughput but also less dynamic behavior. This dynamic is responseable for a non-termination of the simulations with 100, 200 and 300 sessions. These simulations did not terminate because the confidence level was not achieved before all sessions were worked off. We have plotted the last throughput of each terminated respectively finished simulation in figure 2.3 to get a correlation between number of sessions and throughput. As can be seen in figure 2.3 the curve has approximately the characteristics of an $e^{-1/x}$-function.

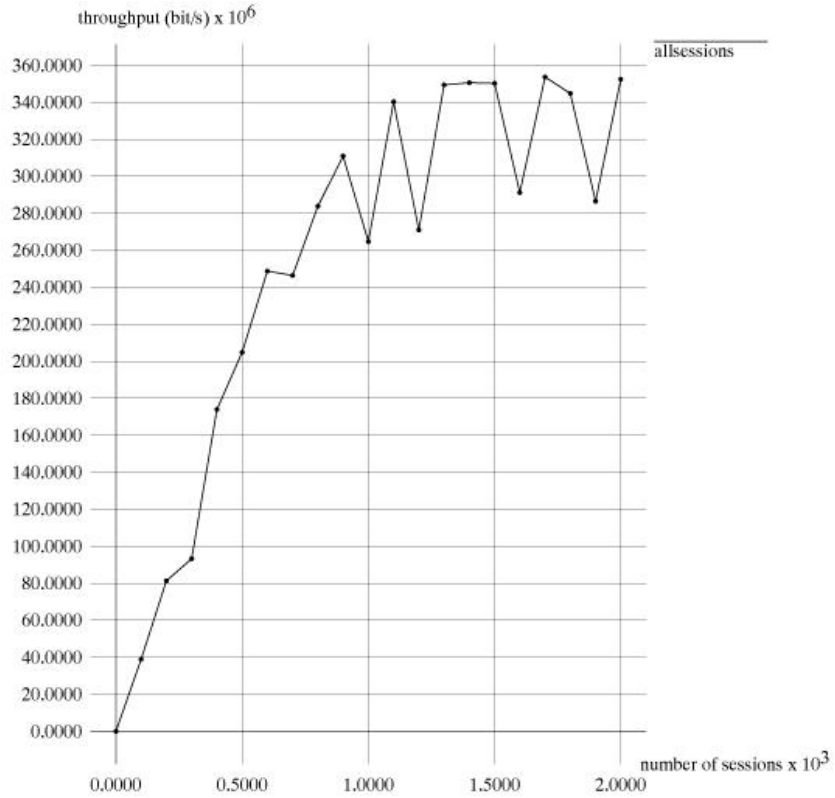

Figure 2.3: last throughput of terminated resp. finished simulations

We recommend a session-number of 1100 or 1300 because there occur high throughputs. Please avoid too high sessions numbers (of 1600 and above) because they result in very high memory usage. Therefore the smallest possible session-number should be chosen! The ripple in figure 2.3 for more than 900 sessions appear because of two observed parameters for termination: throughput and delay for Macs 0 and 5. Both must lie in the defined range for termination. Some simulations terminate earlier while others require more time. The earlier terminated simulations result in lower throughputs (e.g. 1000, 1200 and 1600 sessions) because the delay has reached a steady value earlier, too. In these cases the last throughput-value (which is plotted in figure 2.3) is a little lower than the value of others. To assure that the WDM network operates at full capacity we have plotted the link throughputs over the session-numbers (figure 2.4). These values have been determined by Akaroa. The throughputs of 100, 200, and 300 are set to zero because these simulations did not terminate before all sessions have been worked off. Figure 2.4 confirms an optimum number of sessions of 1100 or 1300.

The delays of the queue in the server node -calculated by Akaroa- is plotted in figure 2.5. The delay increases with higher session-numbers, too. This occurs because the load in the network icreases, too.

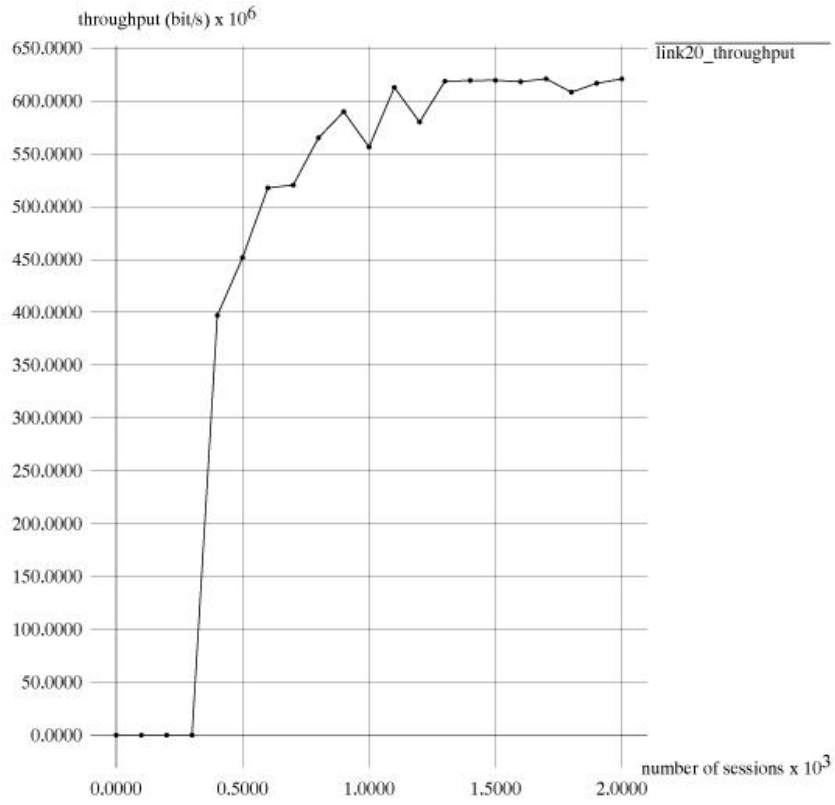

Figure 2.4: throughput of link 2→0

2.2.6 Conclusions

With the parameters described in section "Parameters of the adopted Generator" a high brutto-throughput of more than 600Mbit/s can be achieved. The netto-throughput lies for 1100 sessions at 340Mbit/s. The ratio of netto and brutto throughput for 1100 sessions equals $\frac{340Mbit/s}{613Mbit/s} \approx 0.55$. Thus the Jumbo packets are filled with data packets only up to 55%. Today's backbone networks are used only up to 20% of their capacity. Therefore it seems that the used traffic model produces sufficient load for the WDM network.

By choosing a high number of sessions the dynamic behavior of the first seconds of the simulated time decreases rapidly and plays nearly no role with a session-number greater than 800.

For further tests especially of the Mac-Fairness algorithm distributed client and server topologies on the WDM network have to be simulated.

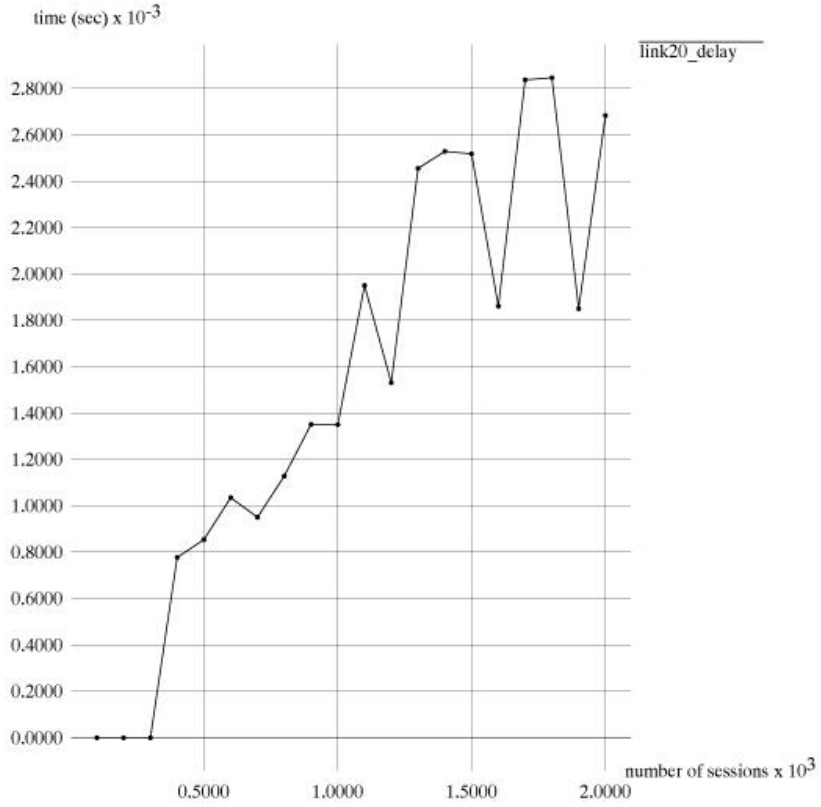

Figure 2.5: delay of link 2→0

2.3 Simulation with several hops in the WDM network between servers and clients

2.3.1 Simulation Goals and Topology

One important issue of the WDM-Mac-Fairness algorithm is the local fairness. "Local fairness" means that two sending nodes, which have to share the capacity of one wavelength, are able to achieve nearly the same throughput. For testing the fairness algorithm, several different traffic flows are necessary through the WDM network. For reaching a good mix of flows we have simulated the WDM network with the topology unidirectional star and ring. Therefore we have connected 600 clients to the WDM nodes 0, 1 and 4. The servers are connected to the WDM nodes 2 and 3. The topology beyond the WDM network is organized as described in Chapter 2.2.1 with parent and intermediate nodes for the servers and parent nodes for the clients. To each client-parent node 200 clients are connected. Each client-parent node itself is connected to one WDM node. The 100 servers of the first simulation are split up, so that one half is connected over parent and intermediate nodes to WDM node 3 while the other half is connected to the WDM node 4. To avoid more confusion in advance clients and parents on WDM node x will be abbreviated as Cx (where x describes the number of

the WDM node) while servers (as well as intermediate and parent nodes) will be denoted as Sx (figure 2.6).

The examination of each Mac on each wavelength is required for testing the WDM network with respect to the local fairness. For this reason, in figure 2.6 the Macs of each node for each wavelength are illustrated with the color of the belonging wavelength. After careful consideration it is obvious that Mac 3 and Mac 8 must not play any role with respect to local fairness in this simulation. Neither Mac 3 nor Mac 8 should send packets on their own because they do not have the shortest path over their wavelength to the S2 and S3.

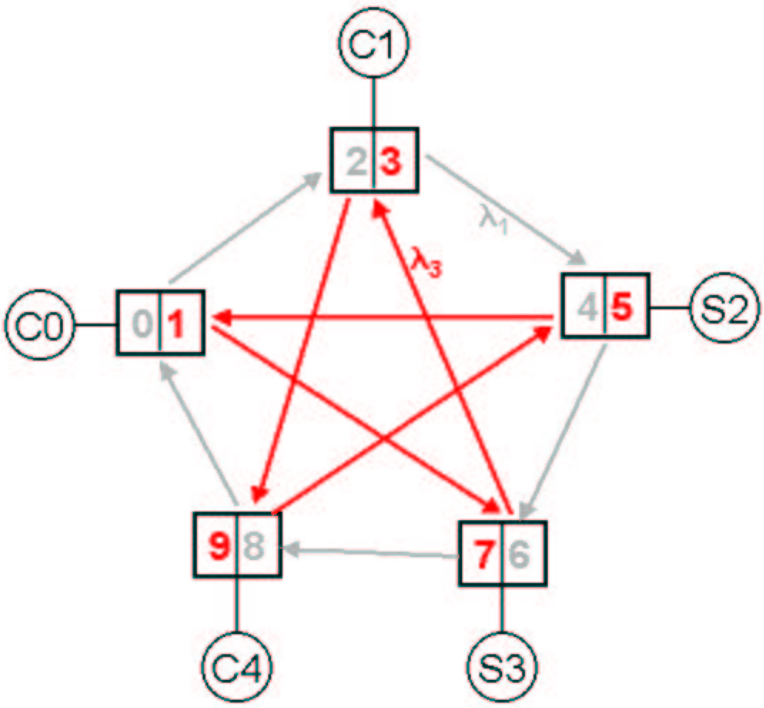

Figure 2.6: WDM, Client and Server Topology

2.3.2 Routing

In the beginning we have used the Dijkstra algorithm for routing (it is the standard static routing scheme of ns2.1b8a). In this case Mac 3 and Mac 8 had some packets to send. So something was really wrong there! The dumped routing table with the next hops shows the reason for this:

node	0	1	2	3	4
0	-	1	1	3	1
1	2	-	2	2	4
2	0	0	-	3	3
3	4	1	1	-	4
4	0	0	2	0	-

With Dijkstra's algorithm node 1 for example has the information that it would reach node 0 (with the next hop over node 2) on the unidirectional ring. But the way over the unidirectional star is shorter (only 3 hops in the star $\Leftrightarrow$ 4 hops on the ring)!

Such errors occured three times in the routing table (marked with red color). Because of unidirectional and different links for the back and forth direction as well as the condition that only one wavelength has to be used to reach the destination node, Dijkstra-Routing can not be used here.

To solve this problem we had to insert all attached nodes of a WDM node in the varp-table of the WDM network. The vitual-arp ("varp") table is a mechanism for the address resolution between nodes, i.e. it provides the next hop information for nodes which want to send in the WDM-network. Routing outside of the WDM network is being done by manual routing. This is not really the best solution but nevertheless it works. A much better solution would be a routing mechanism taking into account the wavelength continuity constraint.

2.3.3 Simulation Structure

Again akaroa is used to ensure the termination of the simulation. To avoid a nonessential memory usage we have chosen a 0.9 confidence level and a precision of 10%. All parameters except for the interpage time have the same values as in the first simulation. We have set the average value of the interpage time from 1 to 0.5 seconds to minimize the dynamic behavior at the beginning. The observed parameters are again throughput and delay of the sending Macs. The simulations with 100, 200, 300, 400 and 600 sessions did not terminate because the sessions were worked off before. In the area of higher session numbers the simulations with 1800 and 2000 sessions consumed too much memory. So they didn't terminate, too.

2.3.4 Results

The delays which occured in the Macs are shown in figure 2.7. The throughput of each Mac is shown in figure 2.8. For the analysis it is important to divide the sending Macs into data-sending Macs and request/ack-sending Macs (the ones with the dotted lines in figure 2.7 and 2.8).

The data-sending and request/ack-sending Macs are listed below to illustrate which Mac sends on the ring (R) or on the star (S):

Mac	data	req/ack
0		R
1		S
2		R
3	-	-
4	R	
5	S	
6	R	
7	S	
8	-	-
9		S

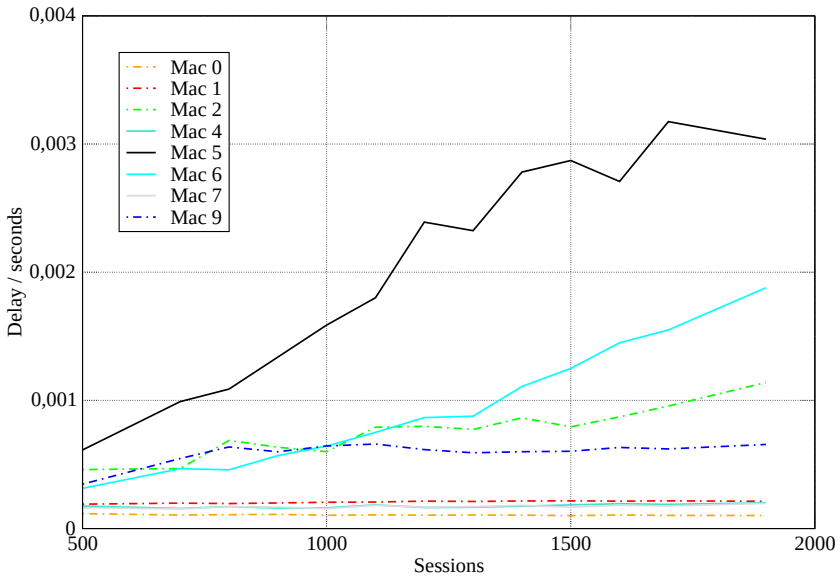

Figure 2.7: delay of Mac-Queues

At first we want to consider the delays: It attracts attention that the delay of Mac5 is the highest and increases with the number of sessions. The delays of Mac6 and Mac2 increase with the number of seesions too, but not as high as the delay of Mac5. Mac9 increases only slowly and keeps constant above 700 sessions.

The others (Mac0, Mac1, Mac4, Mac7) have a small and for all sessions constant delay because they are able to send immediately on their wavelength: Client-Mac0 has to share the ring only with Mac2 for sending requests/acks to S2, Client-Mac1 sends only requests/acks on the star to S3 and has a small delay although Mac5(S2) is sending data packets to Mac3(C1). The server Mac4 is able to send its data packets on the ring to Mac8(C4) nearly immediately; Mac7, sending data packets on the star to Mac3 (C1) and Mac9 (C4), is the only Mac which sends in this part of the star.

The server Mac5, which is sending on the star to Mac1(C0) and Mac3(C0), has a high delay because Mac1 (to Mac7) as well as Mac7 (to Mac3) are also sending on the star. The high load in this part of the network results in the high delay for Mac5.

Mac6 has a higher delay because Mac4 is sending to Mac8 on the ring. Therefore Mac6 has to wait a little. The same happens to Mac2, sending requests to Mac4(S2) and Mac6(S3): Sending requests/acks of Mac0 to Mac4(S2) results in a higher delay for Mac2. Mac9 slows down because Mac5 has to send a lot of data on the star.

The subdivision into data- and request/ack sending Macs is also interesting for the consideration of the throughputs (figure 2.8). Mac0 and Mac1 are sending their requests/acks both for C0 on their respective wavelength. Together they reach (for 1900 sessions) a request/ack throughput of approximately 400Mbit/s. In contrast Mac2(C1) and Mac9(C4) send the whole request/ack traffic of their client nodes. Mac9 sends a lot of requests/acks. Therefore Mac4 must have a high data throughput to Mac8, too. Because of the high request/ack throughput of Mac9, Mac5 is being slowed down. Mac5 provides data for Mac1(C0) and Mac3(C1) with a

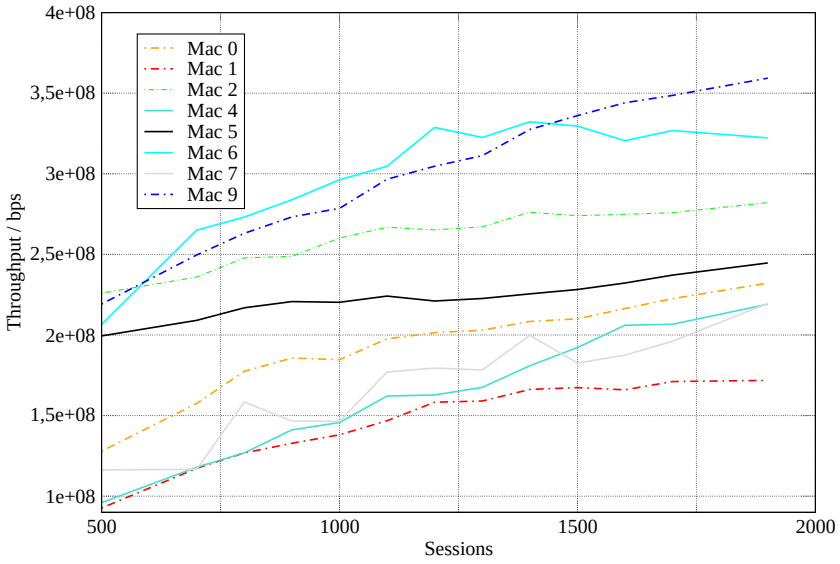

Figure 2.8: Throughput of Macs

throughput of about 245Mbit/s. Mac6 sends data to Mac8 and Mac0, so it has a throughput of 325Mbit/s. Last but not least, Mac7 provides data packets to Mac3(C1) sharing the link with the connection of Mac5 to Mac3.

2.3.5 Conclusions

With the simulation of a distributed client/server model it can be shown that local fairness can be achieved in the WDM-network. As described in the chapter above, the sending nodes on one wavelength (topology) are sharing the whole capacity of the link. So, the Mac-fairness algorithm performs its task.

But there are still some important side-effects: First, there occurs a higher delay if packets from one Mac have to be send to several destination Macs. The reason for this is that for every destination a separate Jumbo packet has to be build. Once sent, Jumbos cannot be repacked before reaching their destination because there it is completely converted the first time from optical to electrical signals. Thus, only packets to one destination can be packed into one Jumbo. If consecutive packets arrive at a WDM node for different destinations, each small data packet results in an own Jumbo packet. For minimizing the delay and for maximizing the netto-throughput, queues for each WDM-destination in each WDM-node should be introduced (virtual output queueing). Then it is possible to collect multiple packets for the same destination before building a Jumbo packet.

The second important thing is the interacting behavior of both topologies. If we have a high data throughput for example on the ring, we will have an increasing load in the star too due to the ack-traffic. This may result in a decreasing throughput for other Macs in the star.

Bibliography

[1] The network simulator ns-2. http://www.isi.edu/nsnam/.

[2] Anja Feldmann and Jörg Wallerich. Design and implementation of a www workload generator for the ns-2 network simulator. http://www.net.uni-sb.de/~jw/nsweb/doku/.

[3] K. Pawlikowski, G. Ewing, and D. McNickle. Project akaroa. http://www.cosc.canterbury.ac.nz/research/RG/net_sim/simulation_group/akaroa/.

[4] Vern Paxson and Sally Floyd. Wide area traffic: the failure of Poisson modeling. *IEEE/ ACM Transactions on Networking*, 3(3):226–244, 1995.

[5] Hagen Woesner, Andreas Köpke, and Eugen Gillich. The ns-2/akaroa-2 project. http://www-tkn.ee.tu-berlin.de/research/ns-2_akaroa-2/ns.html.