94462
COMPUTER COMMUNICATIONS

LABORATORY 3

INTERNETWORKING: FLOW CONTROL ALGORITHMS AND TCP/IP

SEPTEMBER 1998

1 PURPOSE

The purpose of this lab session is to illustrate some of the basic concepts encountered in the TCP/IP protocol suite. TCP/IP is currently the dominant protocol used in internetworking. It has to provide a number of functions to enable successful communications between stations that may be on the opposite sides of the globe, use different transmission speeds or hardware. For example, it defines a common protocol data unit (PDU) such that all stations using TCP/IP have a common base for communication. Furthermore, a unified addressing scheme is defined – the IP addresses – which allow each station on the globe to be uniquely identified. These addresses are used by the routing algorithms to send packets in the right direction. Other important functions include flow control, congestion control and error recovery. TCP/IP uses a number of concepts here, some of which you have already encountered in the lab on ARQ algorithms. However, these algorithms and concepts have to be modified to deal with the complexity of an entire network, not just a single link between two stations.

The objective of this lab is to demonstrate a few of these concepts by use of simulation models in COMNET III. In particular, we are going to focus on

- Nagle's algorithm
- The TCP/IP slow-start algorithm for flow control
- adaptive timeouts to demonstrate the TCP/IP congestion control algorithm

Obviously, TCP/IP provides many more functions. Many of these are beyond the scope of this lab. In fact, some features of TCP/IP cannot even be simulated. They are simply a matter of convention or standard. For example, the definition of the IP addressing scheme has been defined, but there is not much to simulate to demonstrate the performance or the execution of this scheme. You just have to understand the basic standard and be aware how it is used. So, in particular, we will not consider

- the segmentation and re-assembly functions of TCP/IP
- the role of sequence numbers and ordered delivery of frames
- routing algorithms
- network addresses and address resolution protocols
- interaction with higher or lower layer protocols

This lab session builds heavily on the theoretical descriptions of TCP/IP in standard textbooks. To successfully complete this lab session, you are expected to have read such relevant sections on TCP/IP. You are again expected to be familiar with the COMNET III network simulator. To execute this lab, please make sure that COMNET III is installed on your machine and that the model files are also installed under the directory 'C:\CommTut\Lab5'. The following files should appear under this directory:

- NagleA.c3
- NagleB.c3
- SlowStart.c3
- CongCont.c3

In addition, you should find the following subdirectories and files

- Subdirectory: ‘C:\CommTut\Lab5\NagleA’
- File: ‘C:\CommTut\Lab5\NagleA\comments.txt’
- Subdirectory: ‘C:\CommTut\Lab5\NagleB’
- File: ‘C:\CommTut\Lab5\NagleB\comments.txt’
- Subdirectory: ‘C:\CommTut\Lab5\SlowStart’
- File: ‘C:\CommTut\Lab5\SlowStart\comments.txt’
- Subdirectory: ‘C:\CommTut\Lab5\CongCont’
- File: ‘C:\CommTut\Lab5\CongCont\comments.txt’

2 THEORETICAL BACKGROUND

In this section we give you a brief overview of the structure and the main functions of TCP/IP. This overview is by no means exhaustive. The functions and algorithms employed by TCP/IP are far too vast to be described in this short overview. For a detailed treatment of the protocol functions, you should consult a book on TCP/IP, such as [5].

Let us start by positioning TCP/IP within the ISO/OSI reference model. Strictly speaking, TCP/IP are *two* protocols: the transmission control protocol (TCP) and the internet protocol (IP). TCP is a layer 4 protocol - it resides in the transport layer in terms of the ISO/OSI reference model. IP is a layer 3 protocol – it resides in the network layer. Let us describe some of the features of these two protocols in turn.

2.1 OVERVIEW OF TCP

The basic operation of TCP can be summarized as follows: TCP receives messages from the higher layer protocols. These messages are large bit streams. They could represent entire files. TCP segments these messages into TCP protocol data units (PDUs), which are sometimes called *segments*. Each segment is of course given a header. The maximum size of a segment is restricted to 65535 bytes, including the TCP and IP headers. This means that each TCP PDU can carry at most 65495 bytes of user data. If a message is larger than this, the message is segmented into several TCP segments. Otherwise, the TCP segment will be just as long as the message plus the overhead of 20 bytes for TCP.

2.1.1 CONNECTIONS

TCP is a connection-oriented full-duplex protocol. This means that before any data is transmitted between the sender and the receiver, a connection has to be established in both directions. Sometimes these connections are referred to as virtual circuits in other protocols. Effectively, before the data transmission takes place, a connection setup packet is transmitted to the receiver. This packet contains information on the IP address, the port number at the receiver side, the maximum segment size and possibly some user data, for example user passwords. If the connection can be established across the network (i.e. if the receiver does not yet use the specified port), the receiver returns an acknowledgement. After the receipt of this acknowledgement on the sender's side, the connection is established and the data transmission process begins. If the connection can not be established, the receiver returns a reject packet.

Similarly, when the data transmission process has terminated, the connection is torn down. The sender transmits a packet to indicate the end of the transmission. The receiver again acknowledges this message and repeats this process to tear down the connection in the reverse direction. Only when the connection in both directions has terminated is the virtual-circuit released.

2.1.2 TCP HEADER FIELDS

Each TCP segment is given a 20 byte header. The fields in this header regulate the functions of the protocol. For example, 2 bytes are used to specify the destination port. Another 2 bytes are used to identify the source port. 4 bytes of the header are used for the sequence number of the segment, and another 4 bytes are used for the acknowledgement number. We will explain the use of these two fields in more detail in the next section. The remaining 8 bytes of the header are used for error determination (a 2 byte checksum field is used for this), to specify the window size or to transmit control information to the peer, such as the type of PDU. These additional functions allow control messages to be piggy-backed on TCP segments flowing in the reverse direction. Furthermore, they provide some quality of service (QoS) functions, for example by allowing a segment to be marked as 'urgent'. We will not discuss these advanced functions any further in this lab. They are beyond our scope.

2.1.3 FLOW CONTROL

The transmission process in TCP is regulated by a sliding window flow control policy. However, this policy is slightly different from the SRP and GBN algorithms you have seen in the previous lab. In particular, the receiver not only acknowledges any segments received correctly, but it also advertises the available buffer space. This flow control method thus prevents the receiver from being flooded with information.

All acknowledgements or windows are treated as byte streams. Thus, TCP will not acknowledge a certain PDU, but the number of bytes received correctly. Similarly, it will advertise the number of bytes available in the receiver's buffer, rather than the number of PDUs which it would be able to receive.

For example, in figure 1 the TCP layer on the senders side initially receives 2048 bytes from a higher layer for transmission. Assuming the connection to the receiver's side has already been established, a 2048 byte segment is transmitted. Upon successful receipt, the receiver returns an acknowledgement, acknowledging the receipt of the 2048 bytes and advertising that its receive buffer still has 3072 bytes free for more data. The sender then obtains more data from its higher layer protocol for transmission, say 4096 bytes. However, since the advertised window was only 3072, a segment of 3072 bytes is transmitted. Upon receipt of this segment, the receiver's buffer is then full, and the receiver advertises this by sending an acknowledgement with window size 0. Only after the receiver has removed some of the bytes from its buffer, it will advertise again that it now has more space available to receive data. This indicates to the sender that it may now continue to transmit, in case it has received more data from its higher layer protocols.

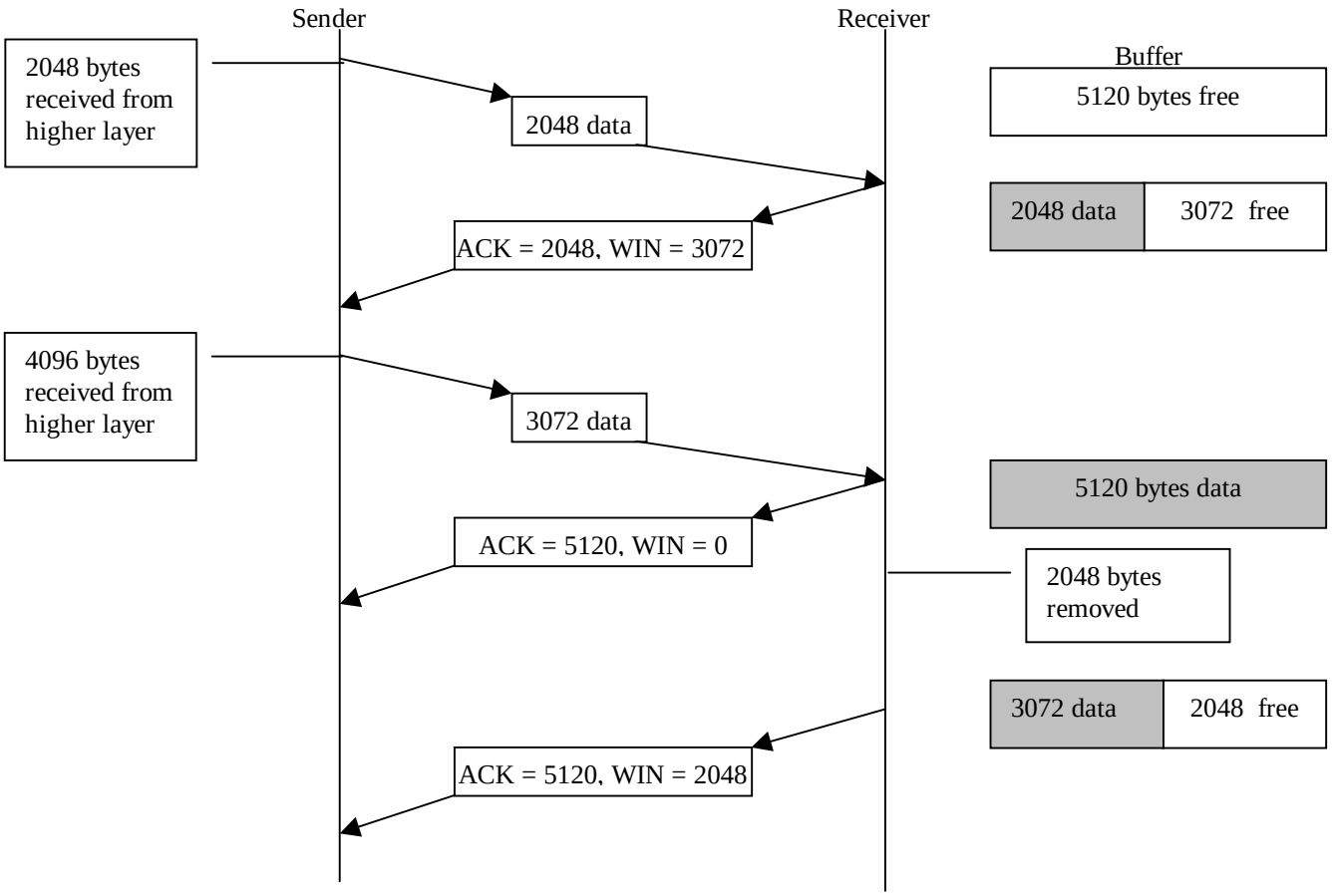


Figure 1: Flow control in TCP

2.1.4 NAGLE'S ALGORITHM

One of the problems faced by TCP/IP is the overhead associated with small user messages. For example, a TELNET application of an interactive editor sends individual characters for transmission. Each character typed on the senders keyboard is segmented into a TCP PDU, adding 20 bytes of overhead. This TCP PDU is then passed down to the IP layer, which adds another 20 bytes of overhead. Thus, a 1-byte message generates a 41-bytes packet for the data-link layer. Furthermore, the receivers TCP immediately acknowledges the receipt of this PDU, sending a minimum of 40 bytes to the sender (assuming no flows in the reverse direction). Then, the editor at the receivers side reads the byte from the buffer, thus generating a window update packet, similar to the one shown in figure 1. This generates another 40-byte packet. And finally, the editor echoes the character back to the sender so that its screen can be updated. Thus, another 41-byte message is sent from the sender to the receiver. This sequence of events is repeated for every 1-byte message generated by the keyboard. To summarize, a 1-byte message generates a flow of 41 bytes in the direction sender-receiver and 121 bytes in the direction receiver-sender.

No doubt that this procedure is very inefficient and wastes a lot of time and bandwidth. To overcome this problem, Nagle's algorithm is used. This algorithms resolves the problem by only sending the first character as outlined in the last paragraph. In the meantime, the remaining characters typed at the sender's side are buffered, until the outstanding character is acknowledged. When this acknowledgement arrives at the sender, hopefully a number of characters have accumulated in the buffer, and they can all be transmitted in a single TCP PDU. Furthermore, on the receiver's side, the acknowledgements are artificially delayed for a pre-specified amount of time. This means that both the window update and the acknowledgement for the character can be piggy-backed onto the echoed-PDU. Using these two schemes, only 2 PDU's flow in each direction, generating much less overhead traffic and thus a more efficient transmission channel.

2.1.5 SILLY-WINDOW SYNDROME

The silly-window syndrome is somewhat related to the above algorithm. Rather than considering the effect of a sender generating 1-byte messages, it considers the effect of the receiver removing data from the buffer in 1-byte chunks. In this case, every time 1-byte is removed from the receiver's buffer, a window update would be transmitted to the sender, indicating that another 1 byte is now free at the receiver's buffer. The sender would transmit a 1-byte message, generating a 41-byte segment. The situation described above would re-occur.

This problem has been addressed by Clark in 1982. His solution is to delay the window update until an appropriate amount of buffer space is available at the receiver's side. Typically, this appropriate amount is defined to be at least 1 maximum PDU size, as determined upon connection establishment. Alternatively, the window update would be transmitted if half of the receiver's buffer is available.

Note that conditions leading to Nagle's algorithm and the silly-window syndrome are related. Similarly, the solutions to both problems are related. Basically, artificial delays are introduced to buffer data and PDUs are only transmitted if it is considered efficient to do so.

2.1.6 ERROR CONTROL

The flow control algorithm above ensures that the receiver is not flooded with data. However, you have to remember that TCP is an end-to-end protocol, and that in between the sender and the receiver lies an entire network with possibly many switches, routers or even satellites. This is where the transport layer flow control and congestion control algorithms differ from the corresponding data-link algorithms. Recall that the error control algorithms in the data link layer rely on a timeout value. At the data link layer, you are only considering 1-hop connections. The delays across this one hop are predictable, if you know the bandwidth capacity of the link, the buffer sizes, and the length of the link. Consequently, the timeout value can be predicted. On and end-to-end basis,

the entire transmission process becomes less predictable. The sender's TCP does not even know the number of hops that the routing algorithm determines to reach the receiver. Furthermore, the delays can now be affected by the intermediary switches. If a switch becomes congested, the end-to-end delay for the TCP segments increases and a timeout may occur. Similarly, the chance of a failure in the connection increases with the number of hops determined by the routing algorithm.

To overcome these difficulties, TCP uses modified concepts of the data link error and congestion control algorithms. First of all, it uses a sliding window algorithm. This means that a new data packet is now transmitted until a previous data packet has been acknowledged. This limits the number of data packets in transmission from the source to the destination. However, because the delay across the network is less predictable, TCP does not use a fixed value for the timeout. Instead, the round-trip-time (RTT) is measured between the sender and the receiver, and it is used to estimate the value of the timeout. In this way, if the network becomes congested, the timeout value is automatically increased, thus preventing timeouts to occur unnecessarily.

Note how important such adaptive timeouts are. If the timeout is fixed and congestion occurs in the network, each packet in transmission would eventually time out. The packets would be re-transmitted, and thus add to the congestion of the network. In a bad case, such a strategy can clog up an entire network!

The algorithm to determine the timeout value in TCP operates as follows. Rather than taking the last measured round-trip time, the timeout is set to

$$TO_{TCP} = RTT + 4D$$
$$D = \alpha D + (1-\alpha)\left|RTT - M\right|$$

where *RTT* is the previously computed value of the round-trip time, *M* is the last measured round-trip delay, and α is a smoothing factor, typically α=7/8. This algorithm for adapting the timeout ensures that some measure of the deviation is also included in the estimation of the timeout, as indicated by |*RTT*-*M*| in the above formula. This factor represents the mean deviation, weighted by a factor of (1-α). It is easier than estimating the actual standard deviation for the round-trip delay.

2.1.7 CONGESTION CONTROL

In addition to the above timeout and flow-control mechanisms, TCP uses an additional window parameter, the congestion window, to take account of congestion within the network. This is different from congestion at the receiver's side, which is treated by the flow control.

The congestion window is modified by the network devices. It indicates the maximum amount of data that the network can transmit. The sender takes both the receiver's window and the congestion window, determines the minimum of the two values and

transmits the respective amount of data. In this way, TCP ensures that neither the network, nor the receiver are flooded with data.

The congestion window is modified as follows: upon connection establishment, the connection window is set to the maximum PDU size for the connection (as determined by the connection establishment process). Each time an ACK returns to the sender before the timer goes off, the congestion window is doubled, thus effectively growing exponentially. This process does not continue indefinitely! Eventually, the congestion window will exceed the receiver's window, in which case it is no longer increased. Alternatively, the congestion window will reach a pre-determined threshold, initialized to 64K. Once this threshold is reached, the congestion window no longer increases exponentially, but only linearly by the maximum PDU size. At this point, the congestion window still grows! Eventually, a timeout occurs in the network. The congestion window will then be re-set to 1 PDU size, and the threshold for exponential growth will be halved from the last acceptable value of the congestion window. The increase repeats itself: initially, the congestion window grows exponentially, until it reaches the threshold. From then onwards, it grows linearly again until a timeout occurs.

The exponential growth phase of the congestion window is called 'slow-start'. Basically, every time a timeout occurs, the congestion window is re-set to 1 and the exponential growth starts again. Figure 2 illustrates the growth of the congestion window. Bear in mind that the sender still takes the minimum of the congestion window and the receiver's window to determine how much data is ultimately transmitted.

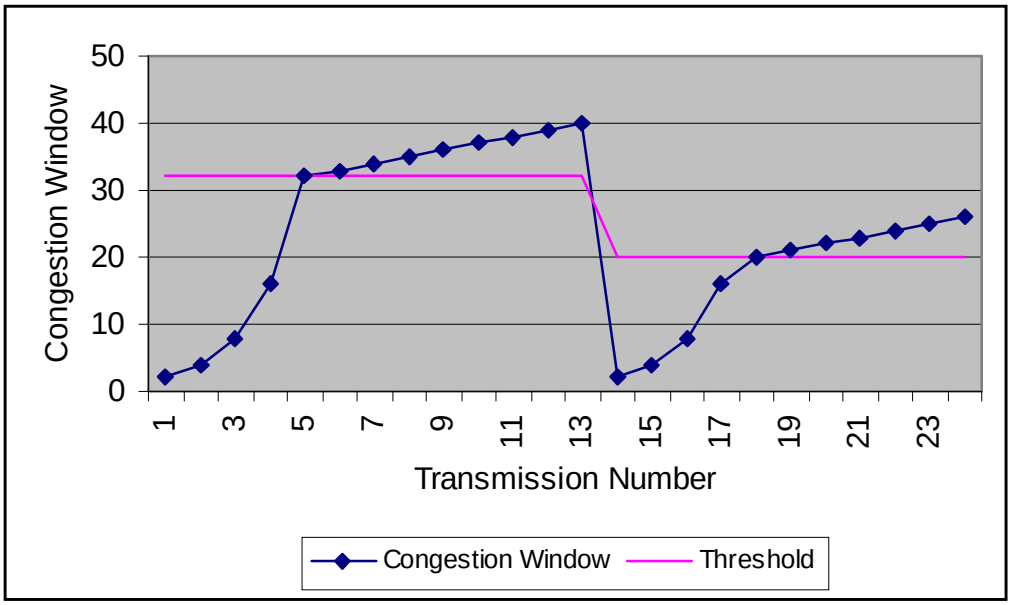


Figure 2: Congestion Window Adaptation

2.2 OVERVIEW OF IP

As mentioned above, internet protocol (IP) resides on layer 3 of the ISO/OSI reference model. Its main task is to route data arriving from the transport layer through to the destination. The main issues in IP are therefor addressing and routing. Since we will demonstrate neither the addressing scheme nor the routing algorithm in this lab, we will keep this section very short.

IP takes a TCP segment and adds its own header of 20 bytes on top of it. So basically, the TCP layer ensures that the data content of its PDU is less than 65495, adds a 20-byte TCP header and then passes the PDU down to the IP layer. At this point, another 20-byte header is added. The main fields of the IP header are

- Source address
- Destination address
- Time-to-live (TTL)
- Header checksum
- Total length
- Type of service

Plus a few other fields which are less important for our purposes. The source and destination addresses are both in the format standardized for IP – the famous IP-addresses. The addresses look like this: 255.255.255.255. This address is inherently hierarchical, sort of like a mailing address. The first few numbers typically represent the network. The last numbers represent the host. For large networks with many hosts, only the first three numbers are used to identify the network. This leaves nine numbers to distinguish between the different hosts. For small networks, the first 9 numbers are used to identify the network, and only the last three numbers are used to differentiate between the hosts. The intermediate solution is also available: six number network identifier and six number host identifier. The Internet can thus accommodate 126 large networks with 16 million hosts each, plus 16382 medium size networks with 65535 hosts, or 2 million small networks with up to 254 hosts. There are, of course, some special address sequences for broadcasts, multicast addresses or control messages. Note that each host on the internet requires a unique address.

The time-to-live (TTL) field in the header is important for network congestion. The routing algorithm used by the internet is local. You cannot have the full view of the entire network. Instead, each packet is sent as a datagram, and a routing decision is made at each router in the network. The routers inform each other about the available paths and the congestion in the network by routing table updates. However, this leaves the possibility of a datagram being passed around in circles between routers. Without the TTL field, an IP packet may remain in the network indefinitely, thus adding to the network congestion. Effectively, this field is decreased as the packet remains in the network, and as soon as this timer expires, the packet is destroyed. Note that this only happens if the packet has not yet reached the destination.
The remaining fields are self-explanatory. The header checksum allows each router to determine whether the header has been corrupted during transmission. Note that only the

header is checked. This prevents routing errors, for example when the addresses become corrupted. The total length field indicates the length of the IP datagram. This is used to properly handle the datagram. Finally, the type of service field allows IP to implement simple QoS functions, for reliability or speed.

IP uses the Open Shortest Path First (OSPF) or the RIP minimum hop (based on Bellman-Ford) routing algorithms. OSPF works on a number of distance metrics and it acts dynamically on these. The algorithm represents the network as a directed graph, in which each arc is assigned a metric. This metric may be based on the distance, the delay or any other metric. OSPF then determines the weighted shortest path between the source and the destination. This algorithm is made possible by routers exchanging routing table updates periodically (link state updates), informing their neighbors about the state of the network. Eventually, these table updates propagate through the network to inform all the routers about the availability of the links and the level of congestion. Based on the propagated information, the routing decision is then made.

3 EXPERIMENT 1

The first experiment in this lab will demonstrate the event sequence and the inefficiencies occurring during a TELNET TCP/IP transmission. You will see how inefficient the transmission of single characters over TCP/IP is and how Nagle's algorithm improves the transmission.

The model is depicted in figure 1.

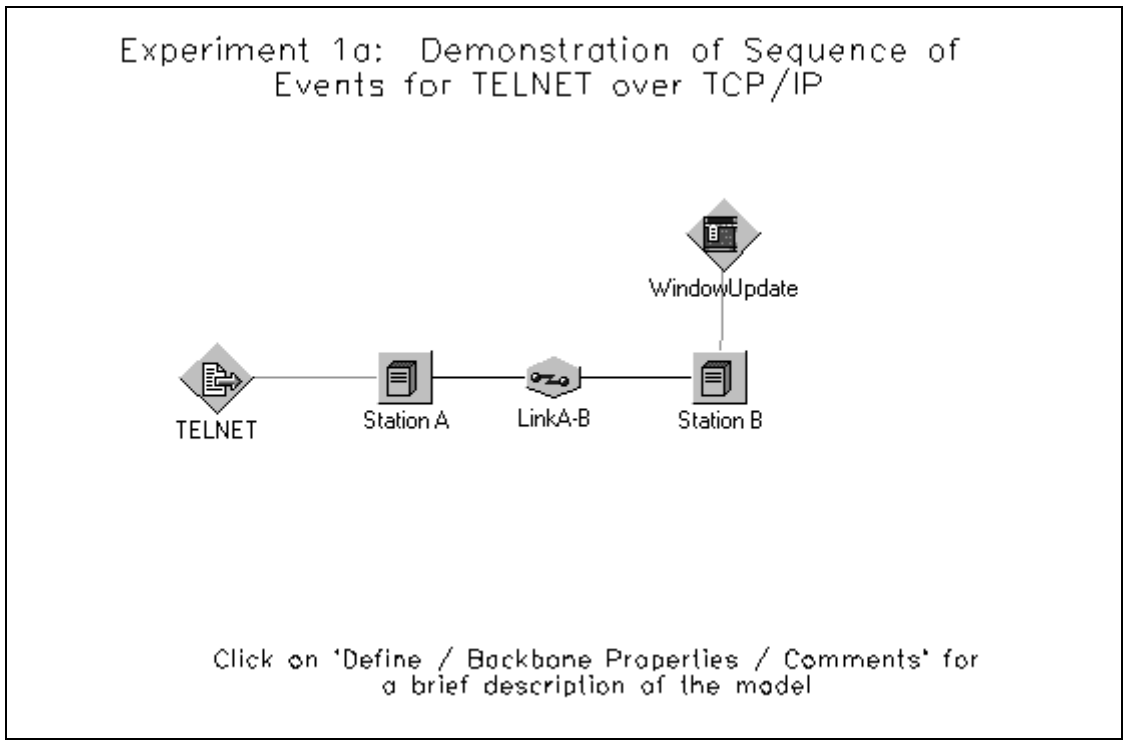


Figure 3: Model Layout for experiment 1

The model is once again very simple. It demonstrates the sequence of event described under section 2.1.4. Station A is assumed to be a terminal running an interactive editor remotely using TCP/IP. Station B represents the mainframe where the editor runs. The user types a single character, represented by the message source 'TELNET' at station A. This character is then transmitted over the link 'LinkA-B' to the mainframe, station B. Upon arrival, the mainframe executes the command sequence hidden behind the application source 'Window Update'. This application source first of all transmits a 40-byte acknowledgement, then it simulates the process of the editor reading the buffer. Following this operation, the window update is transmitted to station A. The character is then processed by the mainframe and finally the character is echoed back to station A.

Let us describe the COMNET III model building blocks in more detail.

3.1 NODES

The two nodes in the model are called 'Station A' and 'Station B', as can be seen in figure 3. Both stations are modeled as default COMNET III processing nodes. This implies that the nodes have infinite buffer capacities and that the processors are infinitely fast. Note that we are once again not really interested what goes on at the nodes. The nodes simply act as sources and destinations of network traffic. Our goal is to observe how much traffic is generated by this event sequence. We therefore do not collect any statistics on the nodes and therefore retain the default values.

3.2 LINKS

There is only one link in the model: 'LinkA-B'. This link provides the connectivity between station A and station B. It is configured as a 128 Kbps point-to-point link. All other parameters are left at their default values. This means that no propagation delay is modeled. Also, no data-link protocol is modeled. The protocol stack in the simulation is thus determined by the transport protocol, as will be described below. The transport protocol generates packets, which are then transmitted directly by the link, without being further segmented into frames, or without any additional data link overhead added. Recall that a point-to-point link has a first-come-first-server (FCFS, also known as FIFO) MAC policy. This means that all packets are transmitted in the sequence presented by the transport protocol.

3.3 MESSAGE SOURCES

There is only one message source in the model: 'TELNET'. This message source only generates one message at simulation time 0.0. No other messages are generated, since the inter-arrival time is left at the value 'none'. The message has a size of 1 byte. You should verify this by double-clicking on the message source and selecting the tab 'Messages'. The destination of the source is set to 'Random Neighbor'. Since there is only one neighbor in the model, station B, the message is automatically transmitted to station B. Finally, the 1-byte message is sent using the generic transport protocol.

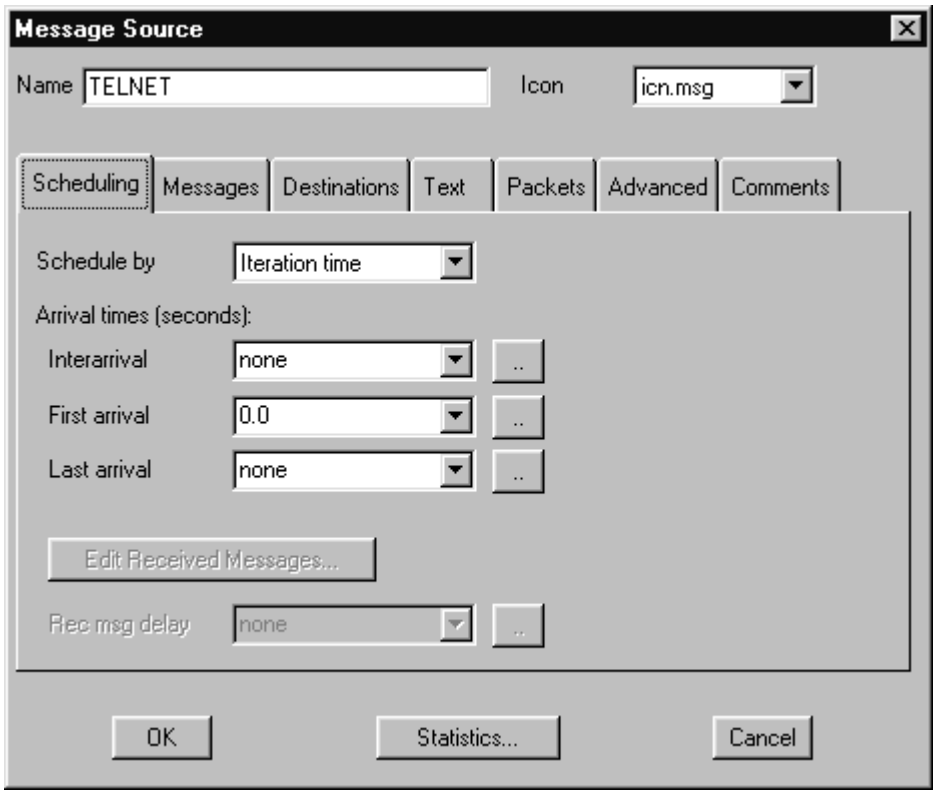


Figure 4: Scheduling parameters for the 1-byte TELNET message

3.4 TRANSPORT PROTOCOL

The transport protocol in this simple model represents a part of the TCP/IP protocol described in section 2. As you can see from figure 5, the maximum data content of a packet is set to 65495 bytes. Each packet is given an overhead of 40 bytes, which represents both the IP and the TCP headers. The packet identifier, used by COMNET III internally for routing and packet processing purposes at the nodes, is set to IP. This is not a critical parameter, but has been set here for clarification purposes.

Note that the acknowledgement bytes and the acknowledgement priority both take the value 1. We do not really model the acknowledgement process automatically in this model. If you click on the tab 'Flow control', you will see that the algorithm is set to 'none'. This means, that COMNET III will not automatically generate any acknowledgements. The reason for this is very simple: we wanted to demonstrate the sequence of events at the receiver side, and so we are modeling all the acknowledgements explicitly using the application source 'Window Update'. As you will see in the next section, the acknowledgements are modeled using answer commands. In this way, you can trace the model much better, since it is explicitly shown when and what acknowledgements are modeled. This would be much less obvious if we let COMNET III automatically generate the ACKs.

We have made one further simplification in this model: we do not model the connection setup and tear-down events. This is an assumption in this model. We pretend that the virtual circuit has already been established between the sources TCP and the receivers TCP protocol.

To summarize the protocol stack: the message source generates a single byte message. This message is then handled by the transport protocol. Since its maximum packet size is not violated, the 1-byte message is simply taken as a packet, and a 40-byte header is added. This packet is then directly transmitted on the point-to-point link, without any additional segmentation or overhead at the data link layer.

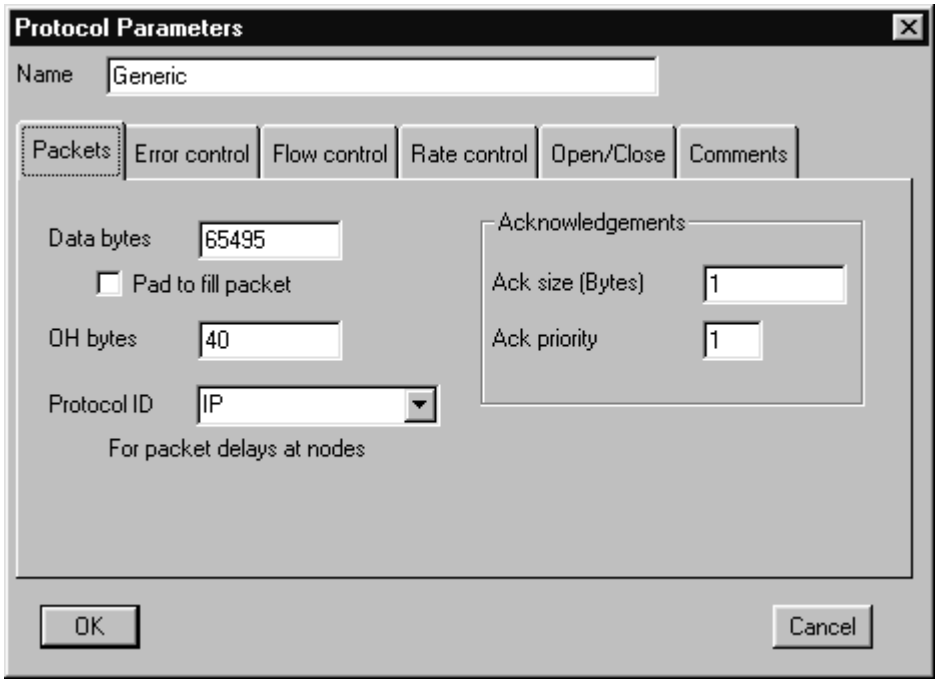


Figure 5: Transport Protocol parameters

3.5 APPLICATION SOURCES AND GLOBAL COMMANDS

The only other source in this model is an application source at station B. You have seen such application sources before in lab 4. They allow you to model event sequences in more detail at a node. In this case, the source is triggered by the arrival of a message, as you can see from the scheduling parameters. The only active button on the tab 'Scheduling' is called 'Edit Received Messages'. If you click on it, you will see that a single message with the text 'TELNET' has to be received to trigger this message. In this way, the message source 'TELNET' in our model activates the application source 'Window Update'.

Once this source has been triggered, the following commands will be executed.

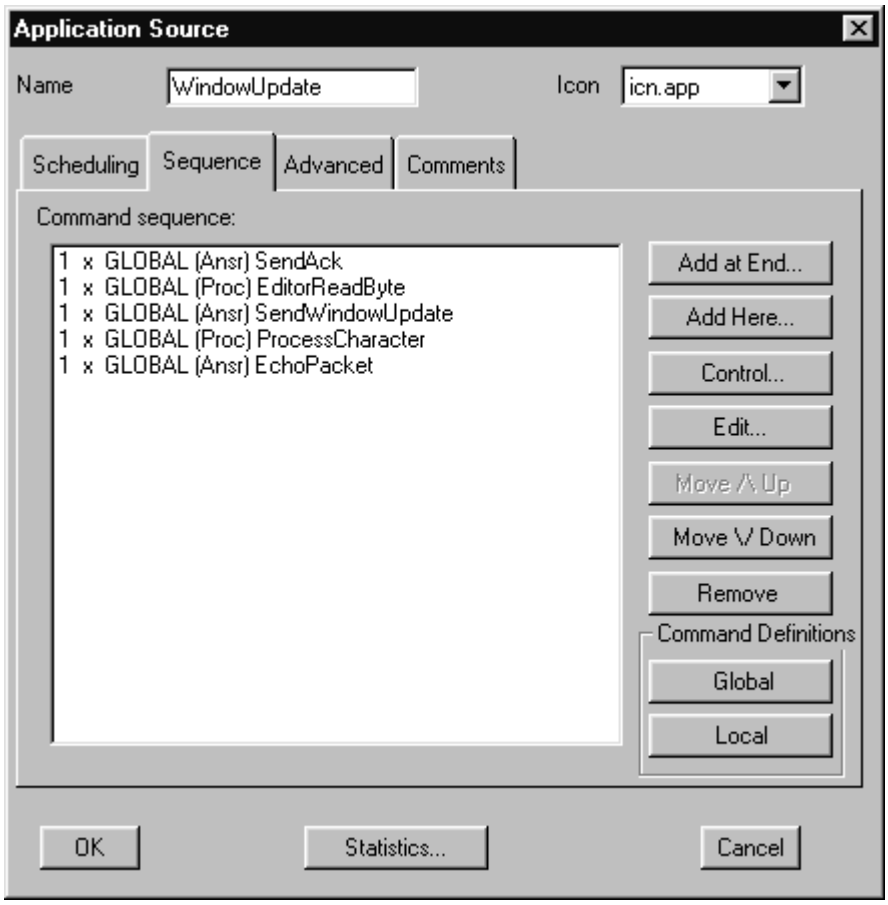


Figure 6: Command sequence in the application source 'Window Update'

First of all, an acknowledgement is generated using a global answer command. The ACK has no data bytes, and so a single packet is returned to station A, which is basically just the packet overhead. The second command models the operation of the editor at station B reading the single byte. It is modeled as a COMNET III processing command. The processing command is modeled arbitrarily as a 5-ms delay, during which the station's CPU is utilized. Following the processing command, another packet is send to station A, representing the window update. Recall that the TCP protocol in station B advertises the available buffer space. Since the single byte has just been read, the station generates this packet to indicate to station A that it may transmit another byte. Again, a COMNET III answer command is used here. In fact, its parameters are identical to the parameters used for the command 'SendAck'. No data bytes are transmitted, only the 40 bytes overhead. Following the 'SendWindowUpdate'-command, another processing delay is incurred at the mainframe station B. This delay models the processing time to process the character. In this case, an arbitrary delay of 10 ms elapses before the CPU is available again and before the last command in the sequence is executed. This last command models the echoing of the 1-byte character to the screen of station A. It is again modeled as a 1-byte message, which follows the generic transport protocol and is thus given a 40-byte packet overhead.

Note that all commands are defined as 'GLOBAL'. This implies that they could be used in any application source attached to any of the processing nodes in the model. You can inspect the details of these commands by clicking on the 'Global'-button, selecting the appropriate command, and then clicking on 'Edit'.

3.6 NAGLE'S ALGORITHM

To demonstrate the difference between the inefficient simulation and Nagle's algorithm, consider the alternative command sequence. This command sequence assumes that the processing delays incurred at station B are less than the delay specified by Nagle's algorithm. As a result, rather than sending the ACK immediately, the acknowledgement is delayed at the receiver. During this delay, the editor reads the single byte from the receiver's buffer. Also, the editor processes the character. So by the time these two processing delays have finished, a single packet can be transmitted back to station A. This packet is effectively the echoing of the character. However, it also carries the piggy-backed acknowledgement for the received packet, and in the window field of the TCP header, the 1-byte receiver window is also indicated. Thus, the transmission only requires a single 4-byte packet, rather than two 40-byte packets and one 41-bytes echoed packet.

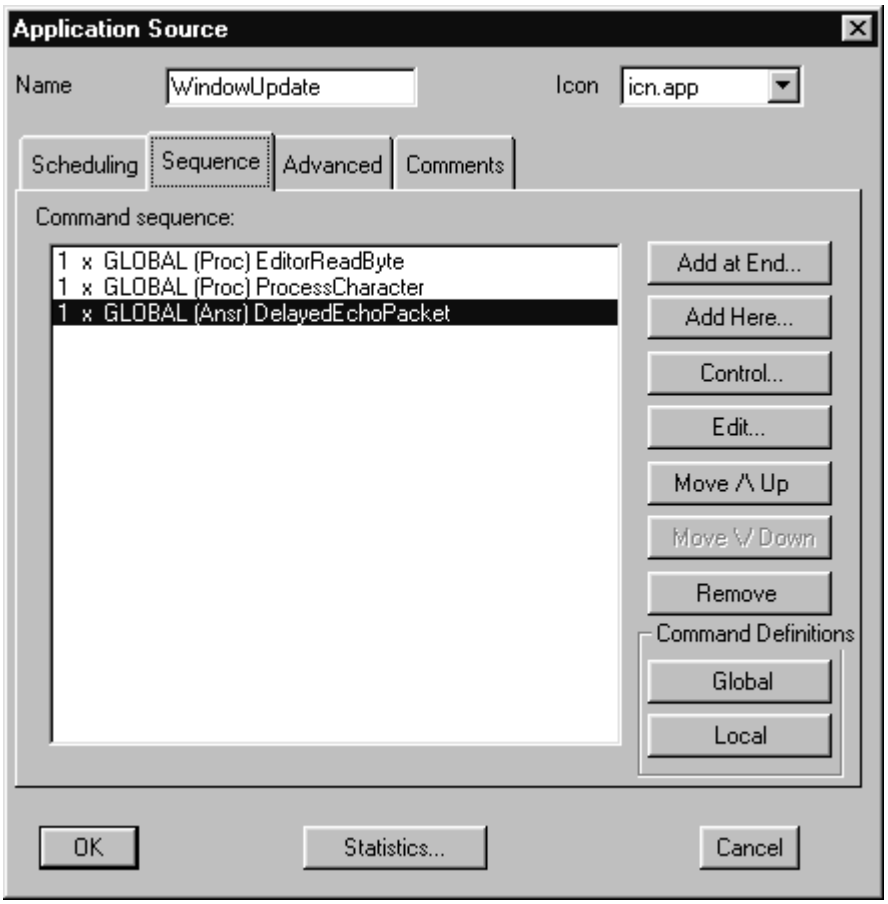


Figure 7: Command sequence for Nagle's algorithm

To Do: Run the simulation of the model 'NagleA' and 'NagleB'. Provide an interpretation of the main results, focussing on the transmission delays and the channel utilization.

Change the inter-arrival time of the message source 'TELNET' from 'none' to 0.01 sec. This simulates the case where a 1-byte message arrives every 10 ms. Run both simulations again and compare the results. Note that we do not assume that the user at station A types 100 characters a second. These values are simply taken for demonstration purposes.

Finally, change the message source 'TELNET' to transmit 2 byte packets every 0.02 seconds. This represents Nagle's algorithm where the user's traffic at station A is buffered until the ACK returns, and then the buffered bytes are transmitted in a single packet. By how much does the link utilization decrease? Does this represent an increase or a decrease in efficiency?

Provide a brief (max. 2 pages, type-written) report with your conclusions of this experiment.

4 EXPERIMENT 2

The second experiment in this lab demonstrates the slow-start function for TCP/IP. The specific purpose of this experiment is to get you to examine the initial seconds when a TCP/IP connection is established and starts to transmit data. In particular, you will be asked to experiment with different values for the flow control window. The model for this experiment is shown in figure 8. The model layout is again kept very simple to make it easier for you to explore all the model building blocks in detail. As you can see from figure 8, the model consists of two processing node building blocks and a new building block, a network device node. The network device node represents a packet switch in this model. Each of the processing nodes are connected to the packet switch through a point-to-point link. There is only one message source generating traffic in the model. It is connected to the source node, and destined for the sink node. Let us describe each of the building blocks in more detail.

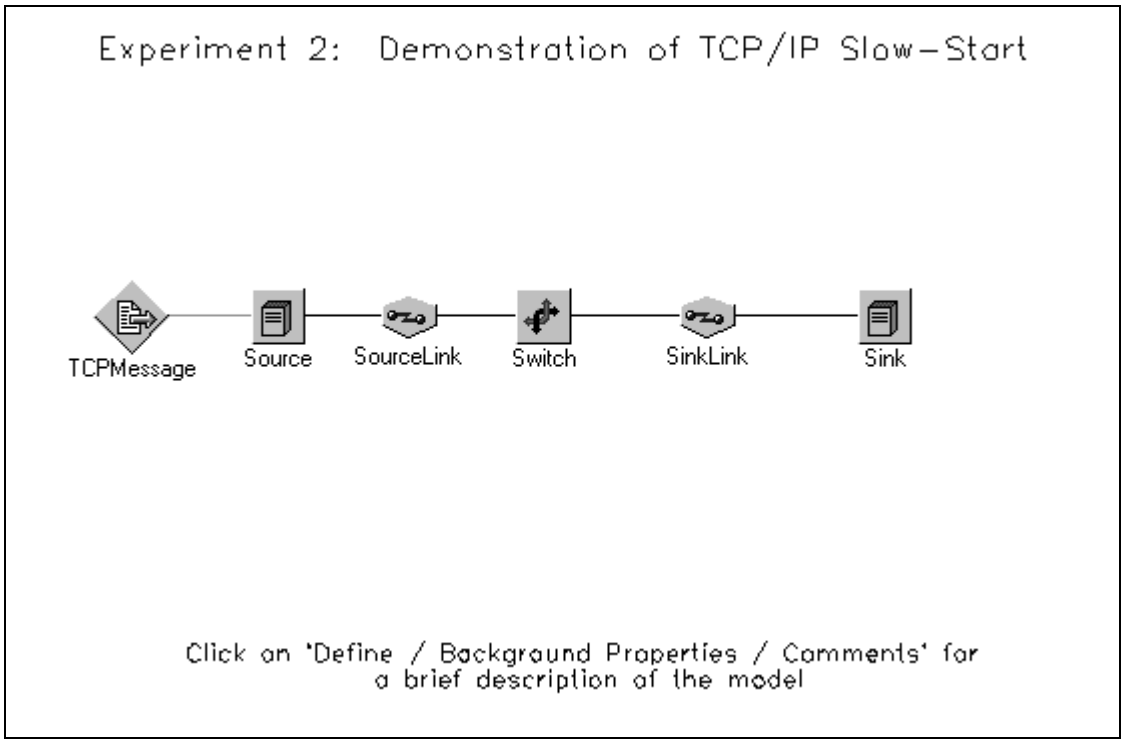


Figure 8: Model Layout for Experiment 2

4.1 NODES

The model consists of two processing nodes and a single COMNET III network device node. The processing nodes are called ‘Source’ and ‘Sink’ respectively. As you can see, the names indicate their function in the simulation. The node ‘Source’ represents the place in the network where traffic originates. Similarly, the node ‘Sink’ represents the place in the network where traffic terminates. Both of these nodes retain their default values

(infinite buffer space, infinitely fast processors), since we are not interested in their statistics.

The third node in this model is a COMNET III network device node. Its internal architecture is depicted in figure 9.

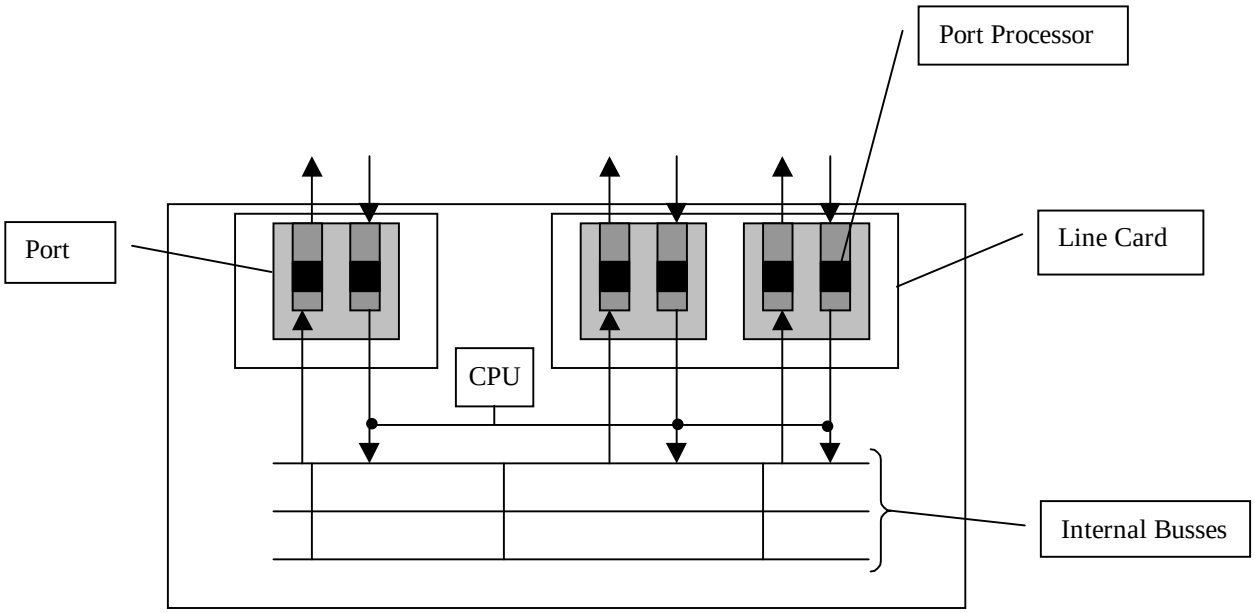


Figure 9: Internal Architecture of a COMNET III Network Device Node

The network device is an extension to the COMNET III processing node. Its similarities are:

- Input and output ports with processors
- A CPU for processing

These building blocks function as you already know from the processing node. In addition, the network device node has a number of busses which connect the ports to the CPU. During a switching operation, a packet not only incurs the queuing delay at the ports and the processing delays at the ports and the CPU. Each packet also incurs a bus delay, which is the time it takes the bus to transmit a packet from the ports to the CPU and vice versa. Note that the bus speed is given in Mbps. It acts like an extremely fast internal link without propagation delay.

The second difference is the line-card. Input and output buffers are physically located on a network card, which is called the 'line card' in COMNET III. When a packet arrives at an input buffer and is destined for an output buffer located on the same link, the packet is not switched through the CPU. Hence, the packet does not incur the bus delay and the CPU delay. Instead, it only incurs the queuing delays at the ports, and the port processing delays. This is how most internet routers function.

For the purpose of this experiment, we will not make use of these functions at all. We simply want to introduce the network device building block here and make you familiar with it. Like with the other nodes in this model, we retain the default values, which are infinite fast CPUs at the ports and inside the switch, as well as infinite buffers. The network device simply acts as a place where packets are switched between point-to-point links.

4.2 LINKS

The two links in the model are again point-to-point links. They connect the source node to the switch, and the switch node to the sink. In this way, they allow packets to travel from the source to the sink.

The bandwidth capacity of both links is set to 512 Kbps. Again, we are not modeling a data-link protocol. We simply create messages which are segmented into packets and immediately transmitted, without any further segmentation or without adding any additional overheads.

4.3 MESSAGE SOURCES

There is only one message source in the model. It is called 'TCPMessage' and only generates a single message, which arrives 0.5 seconds into the simulation. The message size is set to 1000 packets. This is done simply to ensure that a sufficient number of packets are supplied during the simulation time of 2 seconds such that the source never runs out of packets. The destination for the source is set to the 'Random List'-algorithm, which means that the button 'Edit Destination List' is enabled. If you click on this button, you will see that the node 'Sink' is the only destination in the model.

Note that the option 'Random Neighbor' would only transmit the packets to the switch. A neighbor is defined in COMNET III as any node which is one hop away, and so the sink node is not a neighbor. The only neighbor in this case would be the switch.

The packets are set to follow the TCP/IP Sun default parameter set, which we will now describe.

4.4 TRANSPORT PROTOCOL

The transport protocol 'TCP/IP Sun default' defines that each packet has a maximum data content of 1024 bytes. Each packet is given an overhead of 40 bytes, which represents the TCP and IP overheads of 20 bytes each. The size of the acknowledgement is also set to 40 bytes.

The flow control for this transport protocol is set to 'TCP/IP window', which implements the enhanced sliding window flow control algorithm plus the slow-start function which is unique to TCP/IP. In particular, if you click on the '..'-button next to the field

'DEFAULT' under the tab 'Flow Control', and then click on 'Edit' to open up the parameters, you will find parameters to manipulate the adaptive timeout function and the error control function of TCP/IP. Furthermore, the tab 'TCP/IP' notifies you that for this flow control algorithm the slow-start function is indeed implemented, but that it needs no parameters for you to set.

For this experiment, we have disabled the adaptive timeout function of the flow control algorithm. The reason is that we want to focus on the demonstration of the slow-start function, and thus we do not want any other function to distort the demonstration. For this reason you will see the parameter 'Retransmission timeout' under the tab 'Adaptive Timeout' set to the value 'Constant'. All other parameters on the tab then become obsolete.

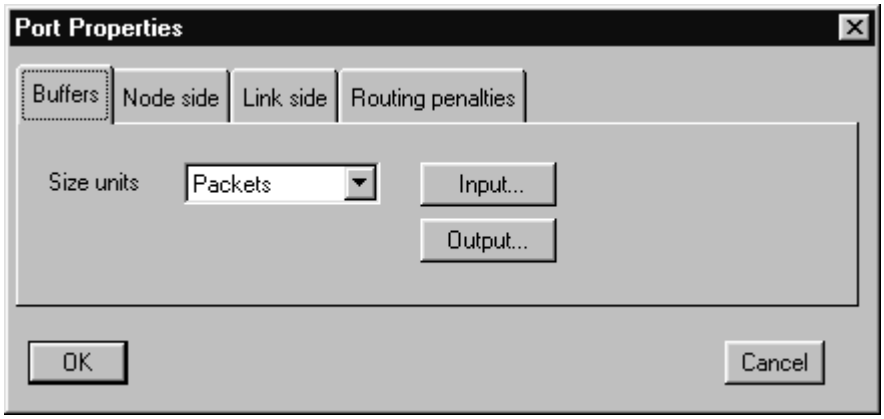
Since we are not modeling limited buffers nor any link errors in this experiment, a packet will never become corrupted. The parameters under the tab 'Error control' are thus not important. If you are really interested, you may want to check the COMNET III manual for a description of the fast-recovery error control algorithm.

To Do: The goal of this experiment is for you to see the impact of the TCP/IP slow-start function. For this reason, you are required to run the model under the following scenarios

- Select the option 'Sliding Window (Enhanced)' for the flow control tab for the transport protocol. This requires you to click on the menu 'Define / Protocols', to edit the transport protocols, to edit the protocol 'TCP/IP Sun default', to click on the tab 'Flow Control', and then to select 'Sliding Window (Enhanced)' as the flow control window algorithm. Set the window size to 16.
  - You can only observe the operation of the windowing algorithm by plotting the level of the output port buffer at the source node. Since all the packets are dumped into this buffer instantly when they are created, this level indicates how the window algorithm sends packets. You should verify that the real-time window for the output port buffer at the node 'Source' is activated. To do so, click on the arc connecting the node 'Source' with the link 'SourceLink'. You will see the dialog box depicted in figure 10. If you click on the button 'Output' and then on the button 'Statistics', you will see that the option 'Real-time Port Buffer Level (ON)' is set. This will automatically show you the level of this particular buffer while the simulation is running.
  - Run the simulation for 2 seconds.
- Change the flow control algorithm for the transport protocol back to 'TCP/IP window'. You should follow the same steps as outlined above to do this.
  - Make sure that the window size is again set to 16

- Run the simulation for 2 seconds.
- What can you observe? Can you find any interesting statistics in the reports to support your observations?

- Now run the same simulation for 2 seconds with values for the window size of 2, 4, 8, 32 and 64.



- Observe how the port buffer level builds up.

Figure 10: Properties of the output port of link 'SourceLink' at the node 'Source'

Provide a brief (max. 1 page, type-written) report, summarizing your findings of this experiment.

5 EXPERIMENT 3

This experiment is simply an extension of the last one. We are now going to increase the simulation time and introduce some congestion in the network. You will see how adaptive the TCP/IP flow control algorithm is and how the flow control window varies as congestion builds up or decreases.

As you can see from figure 11, which shows the model layout, the network topology is identical to the previous experiment. However, some of the parameters for the nodes have now been changed such that congestion builds up. Let us describe these changes in detail.

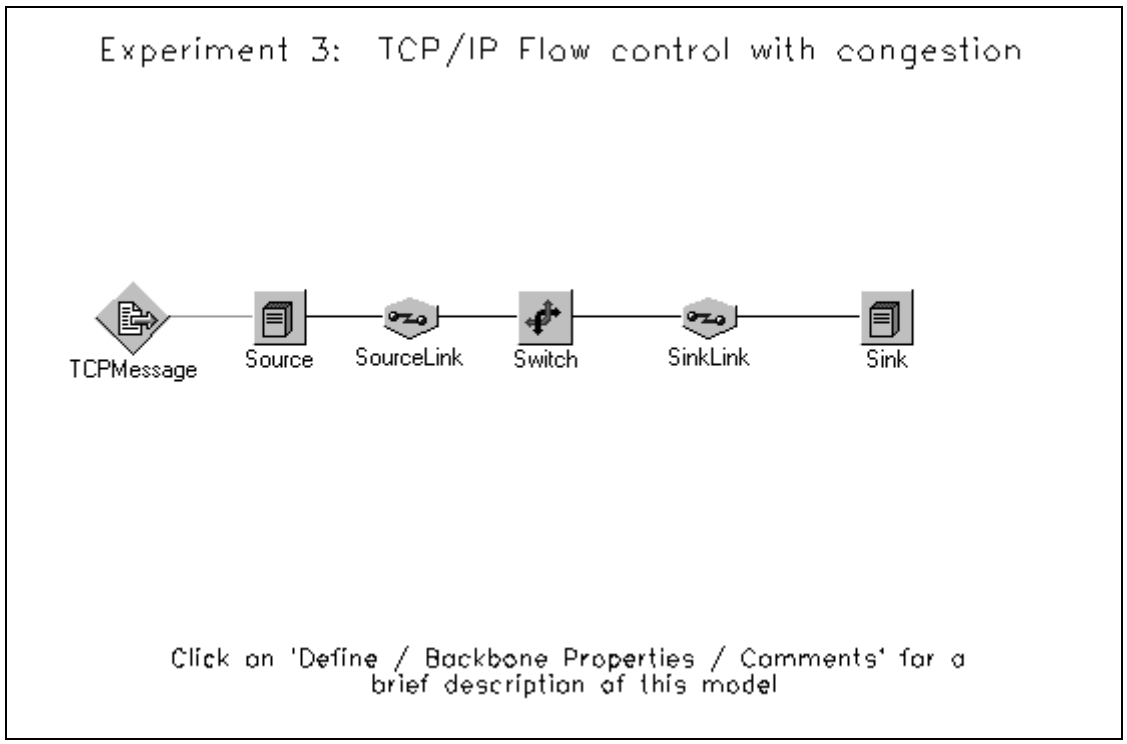


Figure 11: Model layout for experiment 3

5.1.1 NODES

The parameters for the node ‘Source’ do not change at all. The other two nodes are given their own parameter sets, which take on the name of the node.

For example, the node ‘Sink’ now also has a parameter set called ‘Sink’. This parameter set defines a packet processing delay of 10 ms per packet. You can verify this by double-clicking on the node, then clicking on the ‘..’-button next to the field parameters, editing the parameter set ‘Sink’, and finally clicking on the button ‘Protocol dependent processing times’. This brings up a list of all the protocol IDs used in the model. You should have an entry for IP in the list. Clicking on the ‘Edit’-button again shows you that each packet

with the label 'IP' now incurs a processing delay of 10 ms at this node. All other parameters are left at their default values.

Similarly, the node 'Switch' now has a parameter set called 'Switch'. This parameter set also defines a processing delay for IP packets, which has been set arbitrarily to 20 ms per packet. You should follow the steps described in the last paragraph to verify this setting.

5.1.2 LINKS

The link parameters have been given the a higher link speed of 51200 Kbps. This ensures that the packets are almost instantly dumped into the input buffer of the switch, which is intended to be the bottleneck in the simulation.

5.1.3 PORTS

Only one port has been given particular parameters. It is the input port to the switch from the link 'SourceLink'. You can get to it by double-clicking on the arc between the link icon for the 'SourceLink' and the node 'Switch'. A dialog box as in figure 10 will appear on the screen. If you click on the button 'Input…', you will see that the buffer size has been limited to 10 packets. This value has been chosen because the flow control window is set to 16. It is therefore quite likely that this buffer will become congested!

5.1.4 MESSAGE SOURCES

The message size has been changed to only 500 packets.

5.1.5 TRANSPORT PROTOCOL

The transport protocol now implements all the functions of the TCP/IP protocol. The adaptive timeout parameters under the tab 'Adaptive Timeout' now use the algorithm 'RTT+M*Deviation'. However, this algorithm will be the focus of this experiment. You will be asked to change the algorithm and observe the differences in the transmission window that results from these changes. Recall that you have to click on the '..'-button next to the field 'Parameters' when you are in the main transport protocol dialog box, and then click on 'Edit' to access the enhanced transport protocol parameters.

Through this experiment, we will assume that the window size is initialized to 16 packets.

The error control parameters still remain the same.

To Do: Run the simulation for 30 seconds for the following algorithms

- 'RTT+M*Deviation'
- 'RTT*M'
- 'Sliding Window (Enhanced)' with default parameters (this effectively disables the slow-start function)

- ‘Sliding Window’
- no flow control

Comment on the message transfer delays (i.e. the time to completely send a 500 packet message from the source to the sink) and on the losses in a brief (max. 1 page, type-written) report.

6 DELIVERABLES:

Experiment 1:
- Print-out the reports generated by the simulation.
- Provide a screen-shot of your model.
- Report

Experiment 2:
- Print-out of the reports.
- Provide a screen-shot of your model.
- Report.

Experiment 3:
- Print-out of the reports.
- Provide a screen-shot of your model.
- Report.

Note: All reports have to be in letter format. All interpretations have to be typed.

Marking Scheme:

Interpretations: Experiment 1 - 35%
Experiment 2 - 35%
Experiment 3 - 15%
Format: 15% (this includes completeness, clarity, form)

END OF LABORATORY 3