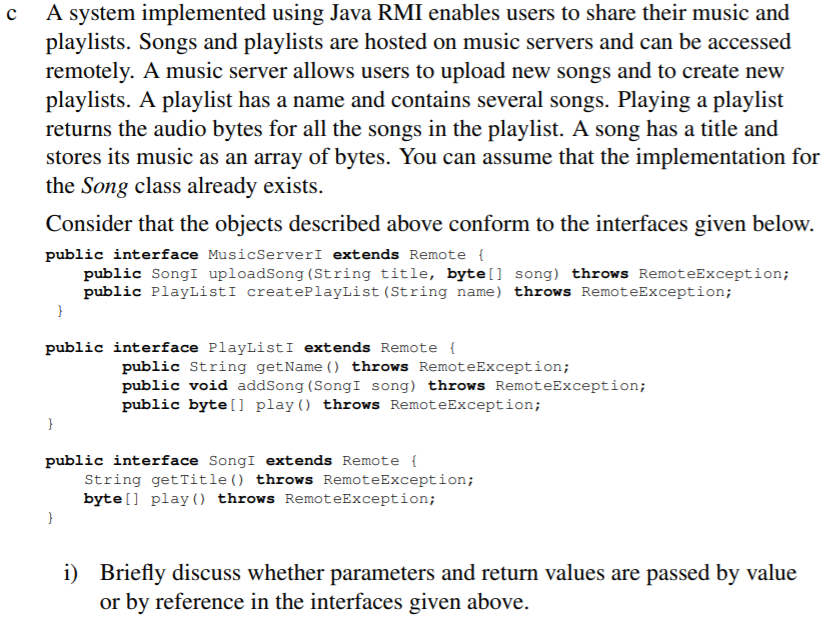


Basically, stubs are required for marshalling/unmarshalling of data and for calling communication primitives (send/receive etc.)

When a client calls a remote procedure, it actually calls the stub which packs up (marshals) the parameters and passes them to the transport layer. On the server-side, a dispatcher receives the message and passes it to a appropriate stub (skeleton). The skeleton then unpacks (unmarshalls) the arguments and calls the relevant server process. Once a reply is ready, skeleton marshals the arguments and passes the message to the transport layer. Client stub then receives the message through the client dispatcher and passes it to the local process (after unmarshalling).



Java RMI uses reflection and generic dispatcher, which eliminates the need for skeletons. When the client stub invokes a remote method, it includes information about the method in the request itself, by creating instances of method class containing types of arguments, return type and exception types. The RMI generic dispatcher then unmarshalls method objects based on this information (also converting remote reference object to local and passing them to the local method invocation).



Local objects are passed to remote procedures by serializing them and passing by value. References to remote objects are also passed by value.

For example, PlayListI.addSong() requires a SongI object, which would be serialized and passed by value.

Same logic applies to return values. For example, PlayList.getName() returns a name of the playlist by serializing the string and passing the value to the remote object that called the method.



Import java.rmi.\*

Public class MusicServer extends UnicastRemoteObject implements MusicServerI {

PlayList serverPlayLists[] = null; //array of playlists on the server

Song serverSongs[] = null; //array of songs on the server

Public MusicServer(Song[] songs, PlayList[] playLists) {

super();

serverPlayLists = playLists;

serverSongs = songs;

}

Public SongI uploadSong(String title, byte[] song) throws RemoteException {

//create a song and add it to the serverSongs

Song newSong = new Song(title, song);

ServerSongs.append(newSong);

Return newSong;

}

Public PlayListI createPlayList(String name) throws RemoteException {

//create a playlist and add it to the serverPlayLists

PlayList newPlayList = new PlayList(name)

ServerPlayLists.append(newPlayList)

Return newPlayList;

}

}



Import java.rmi.\*

Public class PlayList extends UnicastRemoteObject implements PlayListI {

String title;

Song songs[];

Public PlayList(String playlistTitle, Song[] playlistSongs){

Super()

Title = playlistTitle;

Songs = playlistSongs

}

Public String getName() throws remoteException{

Return title;

}

Public void addSong(SongI song) throws remoteException{

//create a local copy of the remotely passed song

Song newSong = new Song(song.title, song.song);

//add it to the playlist

Songs.append(newSong);

}

Public byte[] play() throws remoteException{

//loop through all the songs and accumulate their bytes

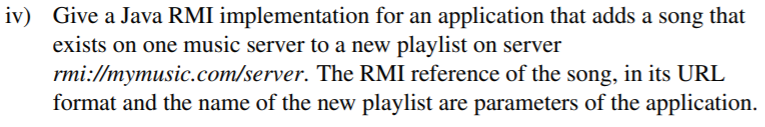
Byte[] allSongs;

For song in Songs:

allSongs.append(song.bytes);

Return allSongs;

}



Import java.rmi.\*

Public void main(String[] args){

//first extract all parameters

String name = args[1];

String songURL = args[2];

//get the security manager

System.getSecurityManager(new SecurityManager());

//get the remote music server

MusicServerI server = null;

Try {

server = (MusicServerI) naming.lookup(“rmi://mymusic.com/server”);

//now add the song

PlayListI newPlayList = server.createPlaylist(name);

//get the song from the song URL

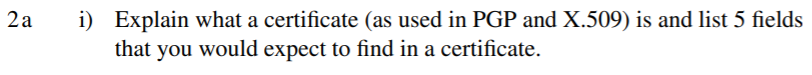
SongI song = (SongI) naming.lookup(songURL);

newPlayList.addSong((SongI) song);

} catch (RemoteException e){

System.out.println(e.getMessage());

}



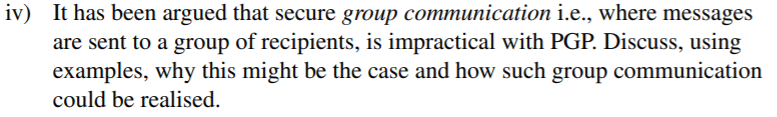
A certificate is a is a signed statement of association between values. Both PGP and X.509 use identity certificates, essentially binding a name to a specific public key. The binding is done by the party issuing a certificate (certificate authority in X.509 or any user in PGP) using their private key to encrypt a number of datafields together. Anyone can then decrypt the certificate using their public key to verify that a specific user (identified by the name) has a specific public key. Once this is verified, this key can be used to communicate with that user, such that only they will be able to decrypt it (using their private key).

Typically, on a certificate one would find the name of the user being certified (e.g. their email address or a domain name etc.), the public key of that user, period of validity of the certificate, the issuer name (and potentially some unique identifier) and a hash of the certificate (done using issuer’s private key) which is used to validate the certificate (anyone can decrypt it using the public key of the issuer).



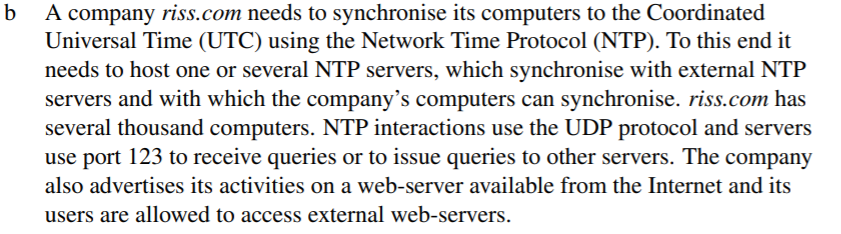
The main difference between PGP and X.509 is that the former is a decentralized system, while the latter is centralized. This influences how *trust* is developed. In X.509, there is a number of Certificate Authorities (CAs). A user of X.509 has explicit *trust* in these authorities (which is derived from the way these authorities are set up, who operates them, how they operate etc.). A user of X.509 then has implicit trust in all parties which are certified by CAs. Notably, one CA can certify another CA, which can then certify another user (Bob). If Alice *trusts* the first CA, she also implicitly *trusts* the second CA and using second CA’s certificate for Bob, also *trusts* Bob.

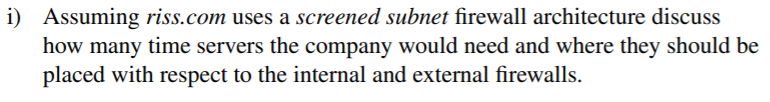
In PGP, there is no explicit trust in a central body (ie no CA), instead users derive trust from other users. Since there is some explicit trust in some of the users you interact with frequently (your *friends*), you can derive further trust from them. This is done by users signing certificates for other users. Hence, if Alice wants to communicate securely with Bob, she can retrieve his certificate which would be signed by a number of different users. Alice may trust some of the users to various extends, and based on that derive the trust in a certificate and hence the public key of Bob.



PGP uses public key cryptography, hence in order to send a message to a number of users, a message for each user would need to be encrypted using their public key. This makes it inefficient, as for N users, each message would require N different encryptions.

On the other hand, symmetric keys allow efficient encryption and decryption. Hence, when establishing secure group communication, first public key cryptography is used to distribute a symmetric session key (generated for that session). This is done by sending the session key (and maybe a nonce) encrypted with each user’s public key (and possibly waiting for them to confirm). Once the session key is established, all communication to all users can be encrypted using this key.





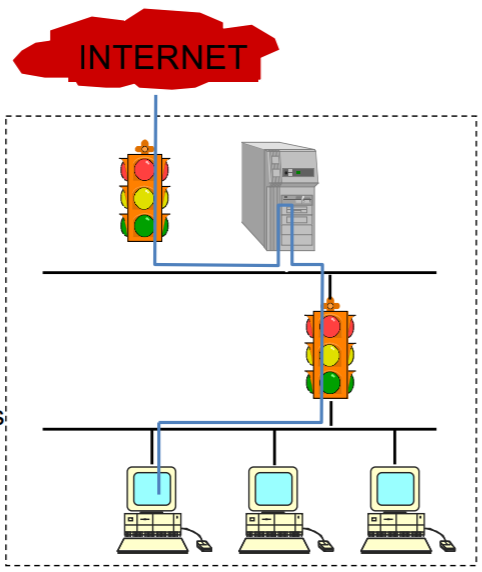
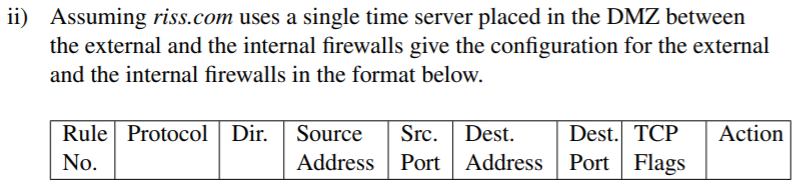


Diagram above shows the schematic of a screened subnet firewall architecture, showing a single bastion host and two packet filtering routers (internal and external firewall). The NTP server would need to synchronize with external NTP servers using a symmetric protocol. The riss.com computers would synchronize their clocks with an internal NTP using a broadcast from the internal NTP server, assuming all computers are on a single high-speed LAN.

**???**

Not sure how many servers though. Maybe have between the internal and external IP filters, so it can communicate with quickly with the external NTP servers (does not need to pass through Bastion). Then another server inside (after the internal filter). This one can synchronize with the previous one in between filters and also since it is on the LAN, it can do efficient and quick multicast to all the computers on the LAN?



Riss.com has a webserver, which must be hosted on the Bastion (port 80), hence must allow external access to Bastion port 80 (in and out) and allow internal access to Bastion port 80 (in and out).

Users are allowed to access external web servers. This is done through a proxy in the Bastion (port > 1024).

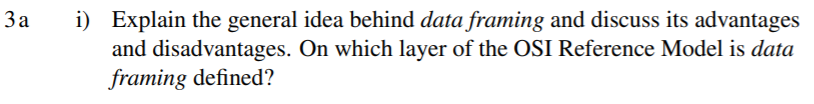
External filter must also allow UDP packets to be sent to the time server on port 123. The time server must also be allowed to send UDP packets back (from port 123). Also internal filter must allow the time server to broadcast the time to internal computers. I assume this is also done through UDP at port 123.

1. Spoofed traffic: someone’s pretending to be inside but is actually outside
2. Allow time server interact with external timeservers
3. Only allow traffic from bastion to the outside (except time server in rule 2)
4. Allow other time servers to send UDP packets to the time servers
5. Allow external users to connect to riss.com web server
6. Allow riss.com webserver to accept connections, but not initiate them.
7. Allow Bastion to access external webservers (port 80) (acting as a proxy for internal computers), as well as initiate TCP connections.
8. Allow external webservers (port 80) to communicate with Bastion proxy, but not initiate TCP connections.
9. Block anything else

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| External IP Filter | | | | | | | | |
| Rule No | Protocol | Dir | Source Address | Src. Port | Dest. Address | Dest. Port | TCP Flags | Action |
| 1 | \* | In | InsideNet | \* | \* | \* | \* | block |
| 2 | UDP | Out | TimeServer | 123 | \* | 123 | \* | allow |
| 3 | \* | Out | Not Bastion | \* | \* | \* | \* | block |
| 4 | UDP | In | \* | 123 | TimeServer | 123 | \* | allow |
| 5 | TCP | In | \* | \* | Bastion | 80 | \* | allow |
| 6 | TCP | Out | Bastion | 80 | \* | \* | ACK | allow |
| 7 | TCP | Out | Bastion | >1024 | \* | 80 | \* | allow |
| 8 | TCP | In | \* | 80 | Bastion | >1024 | ACK | allow |
| 9 | \* | \* | \* | \* | \* | \* | \* | block |

1. Allow time server to send UDP packets into the network for time synchronization
2. Allow response back to the time server using UDP from computers (request to synchronize)
3. Allow access to riss.com webserver through Bastion port 80
4. Allow Bastion port 80 webserver to reply (but not initiate connections)
5. Allow access to external webservers through Bastion’s proxy
6. Allow Bastion’s proxy forward external webserver’s response (but not initiate connections)
7. Block anything else

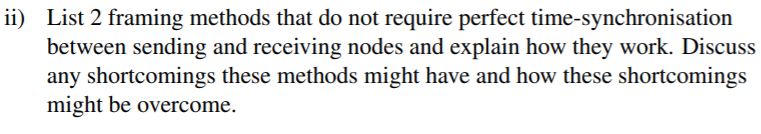
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Internal IP Filter | | | | | | | | |
| Rule No | Protocol | Dir | Source Address | Src. Port | Dest. Address | Dest. Port | TCP Flags | Action |
| 1 | UDP | in | TimeServer | 123 | \* | \* | \* | allow |
| 2 | UDP | out | \* | \* | TimeServer | 123 | \* | allow |
| 3 | TCP | out | \* | \* | Bastion | 80 | \* | allow |
| 4 | TCP | in | Bastion | 80 | \* | \* | ACK | allow |
| 5 | TCP | out | \* | \* | Bastion | >1024 | \* | allow |
| 6 | TCP | in | Bastion | >1024 | \* | \* | ACK | allow |
| 7 | \* | \* | \* | \* | \* | \* | \* | block |



The main idea behind data framing is to create a standard unit of transmission (although data frame lengths vary). This unit of transmission (frame) then can also include additional information regarding the transmission that can facilitate error detection. Moreover, if an error is detected (checksum is not what is expected), it is known that an error is isolated within this specific frame (of course other frames may have errors too), and hence only this frame needs to be retransmitted, instead of the whole communication. Moreover, by including a preamble in the frame, receiver knows exactly when the frame starts and can efficiently extra the payload (actual communication transmitted). Also, by limiting the maximum frame size, collisions are less frequent (as opposed to unbound transmission).

The main disadvantage of framing is the inclusion of additional overhead (preamble, length etc.). While the additional information is useful, it also increases the amount of information that needs to be transmitted.

Data framing is used in the datalink layer.



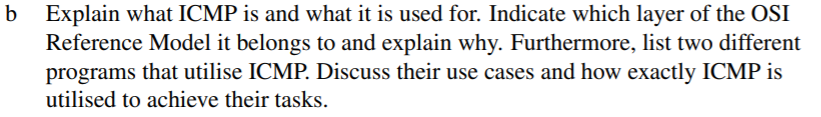
The first, and possibly the simplest, framing method is to include a length field to delimit the frame. For example, before a frame payload, a predefined number of bytes specify the length of the frame. The main issue with this method, is that the length field may be corrupted during transmission, which would result in receiver possibly accepting data that was not part of the transmission.

A better method is to use specific start and end flag and include it before and after the transmission. This way, if the receiver loses track of the transmission, it can locate the next start flag. An issue with using this is that the start/end flag may appear in the actual payload. In order to deal with this, another Esc flag is used around the start/end flag in the payload. Similarly, if Esc flag appears in the payload, it is also encompassed in Esc flags.



Inter-frame gap, amount of idle time between any two consecutive frames sent by the same sender. This ensures that the receiver’s hardware has enough time to recover and prepare to receive the next packet. The IFG depends on the underlying physical layer used for communication (the faster the communication, the less of a gap is required, e.g. for 10 Mbit/s - 9.6 microseconds is required).

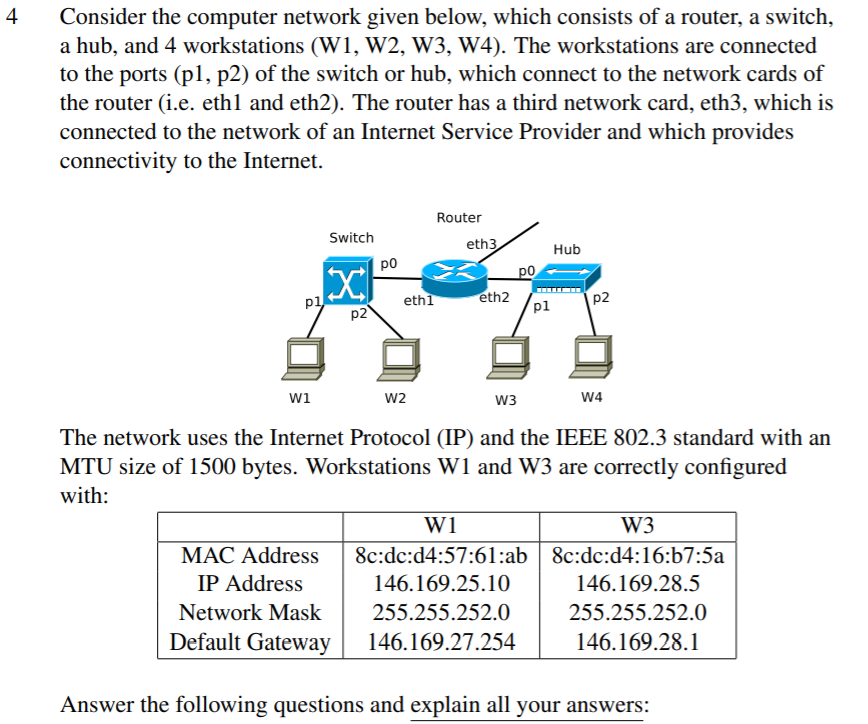
While IFG allows receiver to recover, it can also allow the medium to be taken by another party (since the medium is idle during this time). In burst mode, in order to prevent this from happening, the sender needs to occupy the medium during the IFG.

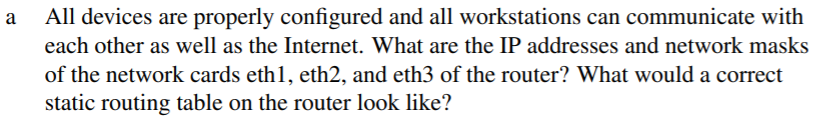


Operation of the Internet is monitored closely by routers. When something unexpected occurs during packet processing at a router, the event is reported to the sender by the ICMP. ICMP can also be used to test the Internet. Each ICMP is carried encapsulated in an IP packet. It also includes its own header which identifies what type of ICMP message it is. While, it may seem that ICMP is a higher layer protocol (since it is inserted into an IP payload), this is misleading as ICMP is critical to proper functioning of the Internet.

One of the common uses of ICMP is the *traceroute* utility. This finds the routers along the path from the host to a destination IP address without any kind of privileged network support. It utilized the ICMP message sent by routers to the sender when the router receives a packet with TtL = 0. The host then sends a series of messages to the destination IP with TtL set to 1, 2, 3 etc. Each packet will time out at successive routers along the path, and each router will send back the *time exceeded* message to the host, identifying itself (the router), effectively tracing the route the packets take to the destination IP.

Another common utility is *ping*. This uses the ICMP type *echo* and *echo reply* messages, and is meant to test whether a specific destination source is online. Sender host sends ICMP *echo* message, and the receiver sends back the *echo reply* message. This utility can also be used to measure the network delay in a communication (by measuring the time delay between messages).





To aide explanation, it may be useful to convert above addresses to their binary form:

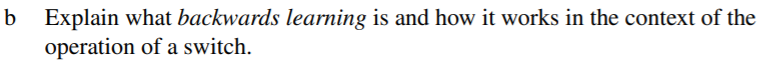
|  |  |  |
| --- | --- | --- |
| W1 | IP Address | **10010010.10101001.000110**01.00001010 |
| Network Mask | 11111111.11111111.11111100.00000000 |
| Default Gateway | **10010010.10101001.000110**11.11111110 |
| W2 | IP Address | **10010010.10101001.000111**00.00000101 |
| Network Mask | 11111111.11111111.11111100.00000000 |
| Default Gateway | **10010010.10101001.000111**00.00000001 |

First thing we take away from this table is the default gateways. For these workstations, the default gateway must be the router, or more specifically the IP address of the router on their network. From this, we know that eth1 IP address is 146.169.27.254 and for eth2 IP address is 146.169.28.1. The network mask for eth1 and eth2 would be the same as for W1 and W2 since they are on the same network (255.255.252.0).

The eth3 of the router is the gateway to the network from the Internet, so it will have its IP as the network ID. Looking at the network mask above and the IP addresses, we can notice that the 22nd bit of the IP addresses of W1 and W2 are different. This would suggest that they are on two differen t subnets, and hence the real (excluding the subnet bit) network mask is 11111111.11111111.11111000.00000000, which is 255.255.248.0. Hence the IP address is 10010010.10101001.00011000.00000000, which is 146.169.24.0 and the mask is 255.255.248.0.

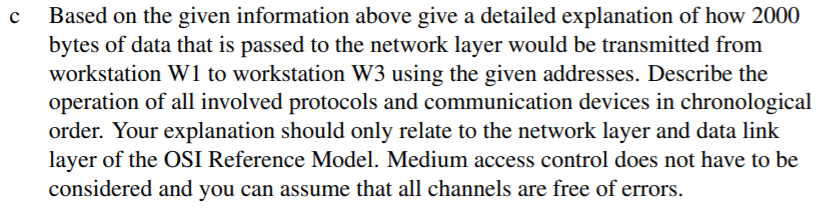
The routing table would then be decided after ANDing the destination IP address with the network mask 255.255.252.0 (network mask of eth1 and eth2 which takes subnets into account) .

|  |  |
| --- | --- |
| IP After ANDing with 255.255.252.0 | Link to Use |
| 146.169.24.0 | eth1 |
| 146.169.28.0 | eth2 |
| default | eth3 |



Switches operate in promiscuous mode, that is, they accept every frame transmitted by the stations attached to each of its ports. It must then decide whether to forward or discard each frame. Switches have a quick lookup hashtables inside them, mapping MACs to port numbers. However, these must be learned.

Backward learning works by inspecting the sender of each frame, and hence mapping that sender (their MAC) to the port from which that frame was received. Since the topologies can change, each entry in the hash table is given a TtL. Periodically, a process in the bridge scans the hash table and purges all entries more than a few minutes old.



Once 2000 bytes of information are passed to the network layer with the destination IP address of W3. The network layer constructs an IP datagram with 2000 byte payload. It specifies a number of fields, such as source and destination IP (v4) addresses, specifies that IPv4 is used, includes the header length and the total length. It may also include the type of protocol that was used by the transport layer (if this is provided). It computes the header checksum and may set some optional fields. Network layer then attempts to pass this datagram to the data link layer, which replies informing that the MTU of the link layer is 1500 bytes (often the network layer would already be aware of this). Now, the network layer must breakup the original datagram into fragments. It chooses two fragments (e.g. 750 bytes long for each excluding header). It then sets the *more fragments* flag in the first fragment to true and false in the second fragment. It specifies the fragment offset for each fragment (0 and 750 bytes respectively). Finally, it must recompute the checksum for each header. Once this is done, it can pass both datagrams down to the datalink layer.

The datalink layer must now construct a 802.11 frame. To do this, first it must find the MAC address where to send this frame. This can be done using a static routing table or just send it to the default gateway (which is the right way in this case anyways). So it now knows the next hop destination MAC address, which is MAC of eth1. It constructs two frames, using eth1 as destination, and fills out standard header fields (checksum, length, source address – MAC of W1, preamble etc.). It then passes it to the physical layer which transports it to eth1 at the router. The router, which operates at the network layer, inspects the payload of the frame, which is an IP datagram. It looks at the destination IP address and uses its routing table to find the MAC address of W3. It inserts the IP datagram back into the frame, recomputes some fields (time to live, destination address = W3 MAC, checksum), and forwards it on to W3. The router does the same with the other frame carrying another fragment.

Finally, the data link layer at W3 receives a frame. It checks whether any errors occurred by calculating the checksum, and then passes the payload to the network layer. Network layer recognizes that this is an IP packet, inspects its header and sees that it is a fragment of an IP datagram. It now waits for the other fragment, and once it arrives (through the data link layer as before), it can reassemble the payloads into a single one using fragment offset fields in the headers. Finally, it passes the reassembled 2000 bytes to its transport layer.