NAME: Ariel Stolerman

CS544 – Computer Networks
Midterm Examination – Spring 12-13
Professor Mike Kain

By turning in this exam, each one of you is explicitly making the following pledge of honesty: "I understand that this exam is to be done individually. The exam is open book (text book and course notes, and lectures – NO other materials may be consulted or used). I have taken this exam individually and have not discussed any question or examined any one else's exam or used any other materials than documented above. I will not copy anyone else's written work, or any copyrighted work (including the text book), nor will I have anyone other than myself prepare any portion of my work. Copying of the solutions of others is expressly forbidden. I will not allow any other person to create, nor to copy, any part of my assignment handed in with my name on it."

The exam must be submitted via Drexel Learn in the allotted time limit – your submission and starting point will be time stamped. Please submit a file (pdf or document) with your last name in the filename (so mine would be kain_midterm.docx)

1. Show all layers of the TCP/IP Basic Reference Model. Describe the basic responsibilities of each layer and name two protocols which exist at each layer. [15 points]

2. Why and how do we break the problem of networking into layers? Why into sublayers? Illustrate this concept by sublayering the transport layer and putting TCP's major features into the appropriate sublayer. [15 points]

3. All layers of the protocol stack share common themes. Why? Illustrate this by taking any protocol that we've talked about in the class (or you've done as your protocol analysis) and show how any five of the common themes exist in this protocol – use examples. [15 points]

4. Define the term "Quality of Service" as we've talked about in class and give two examples of it [10 points]

5. How does a network protocol use a Deterministic Finite Automaton (DFA) to represent the conversation state of a protocol? Why do they have to be deterministic? Sketch out the DFA states of the conversation of a protocol which checks a remote server for updates in a computer program (and installs any changes). [15 points]

6. The sliding window protocol is the heart of many network protocols. Describe ALL of the mechanisms (ACKs, etc.) that were discussed as integral pieces of the SWP. Imagine a network protocol which can stream live video – allowing the user to select between fast play (with possible missed frames) or complete play (guaranteed to show all frames). Would the sliding window protocol be the same in both "modes" of the protocol? Why or why not? Describe the mechanisms (as described in the first part of the answer) that you would see in each mode of this protocol and how they would be used. [20 points]

7. Extensibility is an essential part of a good protocol. Why? Show two examples of ways that good protocols use different ways to be extensible. [10 points]

[optional] Please write any constructive feedback on this course at this time of the semester. How is the class going? How are the teacher and the T.A? How is the book? How is your group going? What is going well? What can be improved? This may also be submitted separately (via email to Mike).

NAME: Ariel Stolerman

1)

Following are the layers of the TCP/IP basic reference model, including their basic responsibilities and examples of protocols at each layer. The TCP/IP reference model is similar to the OSI reference model, where the top 3 layers of the OSI model – application, presentation and session – are encapsulated into one at the TCP/IP model. The layers are as follows:

1. Application layer: this is the user-end top layer of the model, which is in charge of the interface with the user, high-level functionality – what services are provided, session management, what the data looks like (how is it encoded, is it encrypted etc.) In short it is in charge of the data and interaction of users (applications) over the network, where usually application types over this layer include clients (request services) and servers (offer services). This layer connects between the high level user and the top of the network – the transport level (next).
   Two examples of TCP/IP application layer protocols: HTTP and FTP.
2. Transport layer: this layer is in charge of the interface of the application layer – the user – with the network (hardware) underneath. It encapsulates data, either user data or control data, from the application layer into transmittable packets, as the first low-level representation of the data before it is sent to the network. This is the midlayer between the network and the computer. On the network end, it can be seen as the first network-side layer that handles content, i.e. "what" (and not "how" like lower layers). Transport layer security like SSL also takes place at this level.
   Two examples of TCP/IP transport layer protocols: TCP and UDP.
3. Interworking/Network layer: in charge of what links to use between source and destination and makes sure the network works well. It is in charge of pushing the packets onto the network, and routing of packets.
   Two examples of TCP/IP network layer protocols: IP (like IPv4 and IPv6) and IPsec.
4. Data link layer: takes the physical layer and makes it a point-to-point link. The link has certain characteristics, for instance it may be reliable, ordered etc. Often paired with the physical layer to comprise the host-to-network layer or NIC layer.
   Two examples of TCP/IP data link layer: Ethernet, Wireless.
5. Physical layer: defines the actual physical communication on the network – basically defines single representations of "0"s and "1"s and makes sure 0 goes to 0 and 1 goes to 1. As mentioned, this layer is paired with the data link layer, and so the examples of the previous apply here as well.

NAME: Ariel Stolerman

2)

Layering the problem of networking helps breaking down the problem into sub problems, which allow defining solutions and approaches for different parts of the networking problem independently of the others. It allows us to encapsulate common functionalities together, and change or replace layers as to the requirements and needs of the service, a sort of a "mix-and-match" of concrete layers. For instance – UDP vs. TCP approaches for the transport layer; switching between them does not affect the functionality of the other layers (does affect, however, the QoS).

Every layer is defined in general by the protocol at that layer, the service it provides the layer above it and the interface it has with the layer beneath it. The functionality and implementation can be then swapped for each of these tiers, using layering. Sublayering is simply applying the layering concept within a layer. For instance, defining a "wire" as a general layer at the physical layer, with different types as "implementations", like Ethernet or wireless.

Sublayering the transport layer (TCP):

<table>
<tr><td colspan="3">Layer above: application</td></tr>
<tr><td rowspan="3">Transport layer: TCP</td><td>Service</td><td>Handles port numbers, flow control, sending/receiving windows, security</td></tr>
<tr><td>Protocol</td><td>Handles connection, sequencing of data packets, ACK/NACK, retransmissions</td></tr>
<tr><td>Interface</td><td>Handles congestion (flow control)</td></tr>
<tr><td colspan="3">Layer beneath: IP</td></tr>
</table>

NAME: Ariel Stolerman

3)

All layers of the protocol stack share common themes because all protocols regardless the layer they apply to have common problems and properties by which they are measured – same general problems to be solved. These commonalities rise from the fact that all protocols eventually handle some type of conversation. Following are 5 of the common themes we discussed in class, illustrated for the Remote Framebuffer (RFB) protocol, on which I did my protocol analysis paper. In short, the RFB protocol is used for VNC sessions – remote control of windowing systems. It allows clients to remote control basically a desktop on a remote server. The common themes are:

1. Addressing: every protocol has to have an address to talk to another entity in the network with that protocol. Examples of addresses for different layers include HTTP URLs, which have the advantage of being location agnostic; IP addresses – used by many protocols; TCP addresses are basically port numbers. For RFB, the addressing scheme is based on the TCP/IP address: servers use an IP address with port numbers 5900+, allowing several servers to run on the same machine (using port 5901, 5902 and so on). Port 5800 is used for java browser-based RFB clients.
2. Ordered delivery: do messages get in an orderly fashion? For instance UDP has no ordering guaranteed, whereas TCP does – using sequencing and ACK/NACK/retransmission mechanisms. For RFB, ordering at the low-level is taken care of by TCP, however at the RFB level itself – a high level of ordering is maintained – a client can send several requests (e.g. stroking several keys on the keyboard) but receive only one update from the server (an update of the screen, after the client's keystrokes are applied). It is only guaranteed that server updates will not mix chronologically.
3. Segmentation and reassembly: how to take apart and put back together messages? This theme derives from the limitation on message sizes, which require handling this issue. For instance, how to break down a large file and then assemble it back together. For RFB this question applies to how to break down the screen picture and rebuild it at the client side – some approaches include defining "update rectangles" defined by <x,y,w,h> and RGB color; rectangles are then connected in order to build the entire screen picture.
4. PDU (message) definitions: how is a message defined? What are the basic tokens used? What does a message header look like? And how is the delineation of messages handled? For RFB, PDUs are defined by sequences of bytes, usually starting with a byte indicative of the type of message (e.g. client request for server update, client hit a key event etc.), followed by contents (like rectangle update array from the server, in response to a client request). Delineation is determined by either known fixed size of some message types, or varying-size messages with a fixed known position that indicates the size of the message (e.g. size of the array of rectangles for a server update).

NAME: Ariel Stolerman

5. Error detection/correction: how are erroneous messages handles? Is there some correction scheme applied to fix the errors? For instance, TCP uses checksums and NAK transmits to report erroneous messages (garbage) or missed packets. RFB is actually a good example where errors are not taken care of so much; connection errors (like bad authentication) simply close the connection. Errors along the communication may be just ignored (like the server ignoring bad client requests).

Other than the five themes above, the common themes include also connection control, flow control, multiplexing, and the two big themes which are security (trust, authentication, confidentiality, integrity, etc.) and QoS.

4)

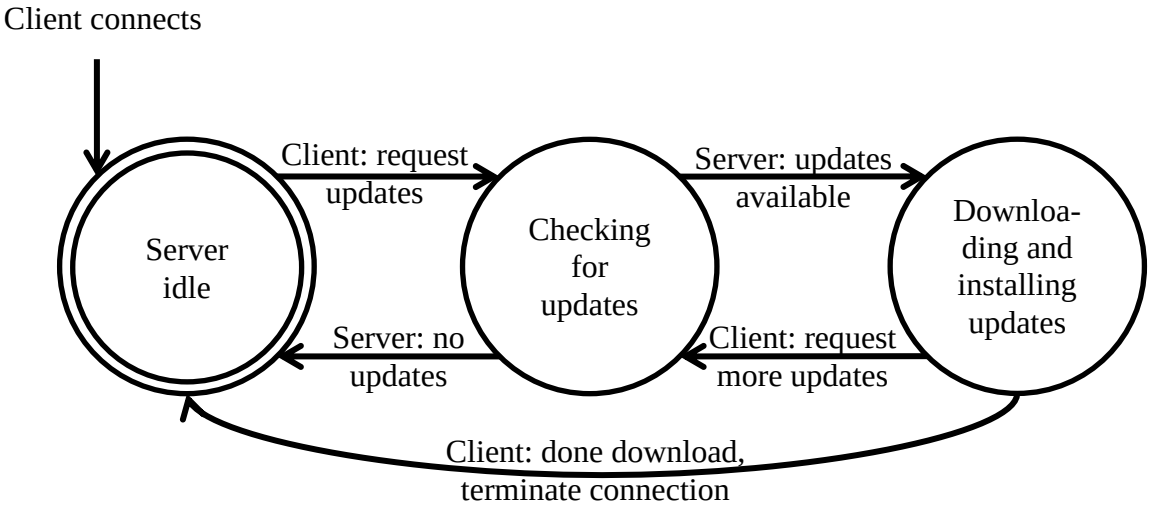
Quality of service is basically what can we get out the protocol – is it easy to use? Fast? Reliable?
QoS refers to all the important characteristics of the protocol that derive from the list of common themes, which eventually define how the service of the protocol is characterized.
Common protocols to illustrate QoS are TCP and UDP – TCP uses mechanisms to ensure a reliable connection, one that we would want to use when transferring files, for instance. The cost of reliability is in speed. UDP, on the other hand does not provide reliability (packets can get lost without being recovered), however it then provides a faster service, one that we would want to use when streaming a video, for instance (where we wouldn't care about a lost frame). Some would say that due to the unreliability of UDP, its QoS is characterized as 0.

5)

Network protocols use DFAs to represent the conversation state of the protocols by defining states of the protocol thru the common actions that are "legal" at those states, which client/server actions (i.e. messages) transferring between states (or from a state to itself). It is basically an illustration of a protocol as a graph of possible states of the protocol, with messages as edges of the graph. The conversation state of the protocol is comprised of the different states along the conversation, defined by what messages are allowed / events can occur at each state and what transitions these messages and events impose.
The DFA has to be deterministic since it corresponds to having a well-defined protocol, with no choices available – at every state of the conversation it guarantees that only a subset of client or server actions is legal/relevant, whereas should it not have been deterministic, it would have created ambiguity and "break" the protocol (actually, theoretically speaking, the class of nondeterministic finite automata – NFAs – is equivalent to the class of DFAs, so any NFA can be translated to a DFA; but the point is that there is no room for ambiguity.)

NAME: Ariel Stolerman

Following is an illustration of a DFA for a protocol that checks a remote server for updates in a computer program, and installs any changes:



6)

Following are all the integral mechanisms a SWP consists of:

- ACK and NAK: acknowledgement and no-acknowledgement messages allow the receiving side of the communication to notify the sending side that it received and did not receive a message. Combined with sequencing and timeout (detailed later), ACK allows notifying the sender that the receiving side received the packet the sender intended to send, or more precisely – the last packet it received. This gives information to the sender what was missed and should be sent again, for ordered and reliable communication. NAK allows simply give an explicit message to the sender that a packet was not received or corrupted. Then the sender doesn't wait for an ACK timeout, but can immediately resend the lost/bad packet, for which it received a NAK.
- Timeout: to increase timing performance in a reliable communication that uses ACK and NAK, timeouts are used to put a limit on: how long should a sender wait for an ACK until it tries to resend the packet, and how long should a receiver wait for a packet it awaits (determined by seq. number) before it sends a NAK on it. This gives some asynchronous properties to the communication (I don't have to wait for an ACK, I can continue assuming the last packet was missed).
- Sequence numbers: used for ordered communication. Each direction of communication (A to B and B to A) has its own sequencing of packets, to allow both sides track what packets are sent and received, what's missing/bad and should be sent again etc. The sequencing in TCP for instance is initialized at

random, so it will be hard to guess (and attackers would have a difficult time attacking the connection, like through reset commands).

- Retransmit: upon an indication of a lost or bad packet transmit, retransmit allows completing the missing data, which provides integrity and reliability of the data and connection (the data will be complete, and correct). Retransmits are applied on either NAK messages or ACK timeouts (did not receive ACK from receiver).
- Windowing (sending and receiving windows): to increase performance, a windowing scheme can be applied where ACK aren't sent after each message, but in bulk. Actually, an ACK would now mean instead of "acknowledging packet #X" simply "give me the next window, starting at #Y". A *sending window* is a window sent by the sender side – i.e. contains data; a *receiving window* is sent by the receiver – i.e. contains an ACK (or NAK), confirming the sent window and requesting the next bulk. Different approaches can be applied for retransmit of missing packets from the window, like GBN or SR. Here in fact the "sliding window" part of SWP comes to manifestation: a window of messages is transmitted, rather than a single message at a time.
- Pipelining: this term refers to having more than one PDU (message) outstanding at any given time. Basically it means that we use windows and cumulative acknowledgments rather than per-packet.
- "Piggybacking": to increase performance, instead of having one sending window and one receiving window, the latter simply "piggyback" the first, i.e. when I send data to the other end, my message will contain both my sequence number and data, and an ACK of the last window I received along with the sequence number of the next window I'd like to receive.
- Checksum: used to sign a message with a checksum of its data, in order for the receiving side to be able to do some integrity check on it, and send a NAK if necessary (when the data turned out to be garbage, inconsistent with the checksum).

Following are the mechanisms as they are to be used/not used in a streaming video application:

- Fast Play: for a fast play, many of the mechanisms above will *definitely* not be used. Those include: ACK/NAK, timeouts, retransmits, piggybacking and pipelining. All those mechanisms are in charge of a reliable communication, which we do not care about for a fast stream. ACK/NAK are redundant since we have no intent to ask acknowledge or request retransmit; timeouts and retransmits are then of course also redundant; piggybacking refers to putting ACK and data messages in one, but ACK is irrelevant here so piggybacking also is; and pipelining is irrelevant because it refers to outstanding PDUs, however here there is no ACK, so no PDU is ever defined as outstanding (i.e. awaits ACK). Some mechanisms may be unused, but using them will give an advantage: sequence numbers and checksums. Those may increase the message size, however can be used on the client side to determine whether the frames it gets are in chronological order (video-wise) and whether they are

corrupted or not. This information can be used to decide whether to present those frames or simply throw them away (no network penalty here other than having longer messages than without seq. numbers and checksums).

The mechanisms that will somewhat be used are windows – we would want to have messages sent at bulks, rather than small messages at a time. The terminology of sending/receiving windows is redundant since we have no "receiving" information to transfer (i.e. tell the server ACK or NAK and such).

To conclude, some of the mechanisms can be adopted to increase reliability if buffering at the application layer is used, which allows some "breathing space" to apply some reliability and still get a fast video stream at the user end. However, for a true stream – all mechanisms should be avoided.

- Complete play: here we would like to use ALL the mechanisms in order to ensure we get ALL the data frames, complete, sound and in-order. We will use ACK/NAK, probably some tradeoff of congestion and fastness for timeout determination, of course retransmit will be used, etc. Basically all mechanisms will be utilized to ensure a complete play.

7)

Extensibility is an essential part of a good protocol (actually, an excellent protocol!) since protocols in nature are incrementally defined, that is, protocols are defined with backwards compatibility (in most cases), which necessitates good grounds for future development upon initial definition of the protocol. Protocols should be planned and defined not only to supply solutions for current problems, but be ready to be extended for possible future applications and developments. A good example of not thinking ahead too far (how could they, really?) is the limited size of TCP fields, which would have probably been implemented otherwise nowadays. To conclude, extensibility is important to allow abstraction that is able to both support current requirements, and provide solid grounds for future developments and newer versions of the protocol, without breaking compatibility with former versions.

Following are two examples that good protocols use different ways to be extensible:

- Version negotiation: part of a good protocol is having a version negotiation at the beginning of the communication, i.e. the initial handshake phase. Versioning allows both ends to 1) make sure they are "on the same page", i.e. support the same actions and "talk the same language", and 2) talk to different endpoints, as long as they support the same "dialect" , for instance a VNC server that supports RFB 3.8 can talk also to VNC clients that support only RFB 3.3 or 3.7.

  This way, future versions can be incrementally applied on top of current supported actions such that at the scope of actions is simply determined at the beginning of a conversation, with respect to the agreed version.

NAME: Ariel Stolerman

- Message type enumeration: some protocols use enumeration of types of messages, or types of any components used by the protocol for that matter, to indicate what action is performed. For instance, the RFB protocol uses different rectangle encodings, indicated by a numeric value, and all encodings are enumerated together. This way, future extensions can simply add new encodings with indices continuing the current enumeration, without harming backward compatibility (same codes are used for the old encodings, new codes are added only after them). This applies to many levels of the message: message type; within a message: component type; within a component: argument type; etc. One important thing in terms of extensibility in that context would be to try and foresee how much space should be allocated for different fields, with look into the future (again: TCP field size example). Larger size means more possibilities of expansion; however there is an obvious tradeoff with message size and therefore performance.