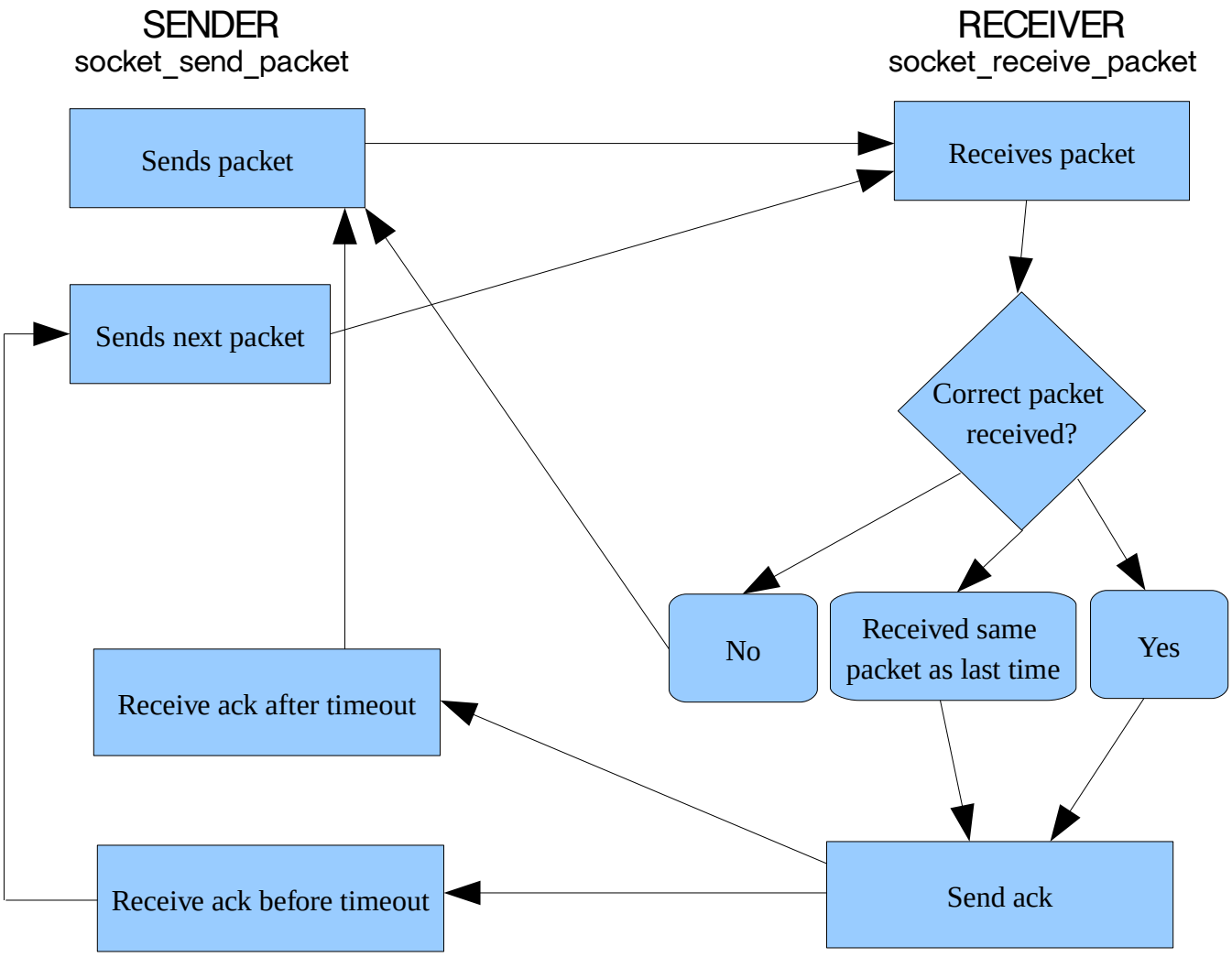
# coms22101 Concurrency and Communications

## Coursework 1

Melissa Weaver

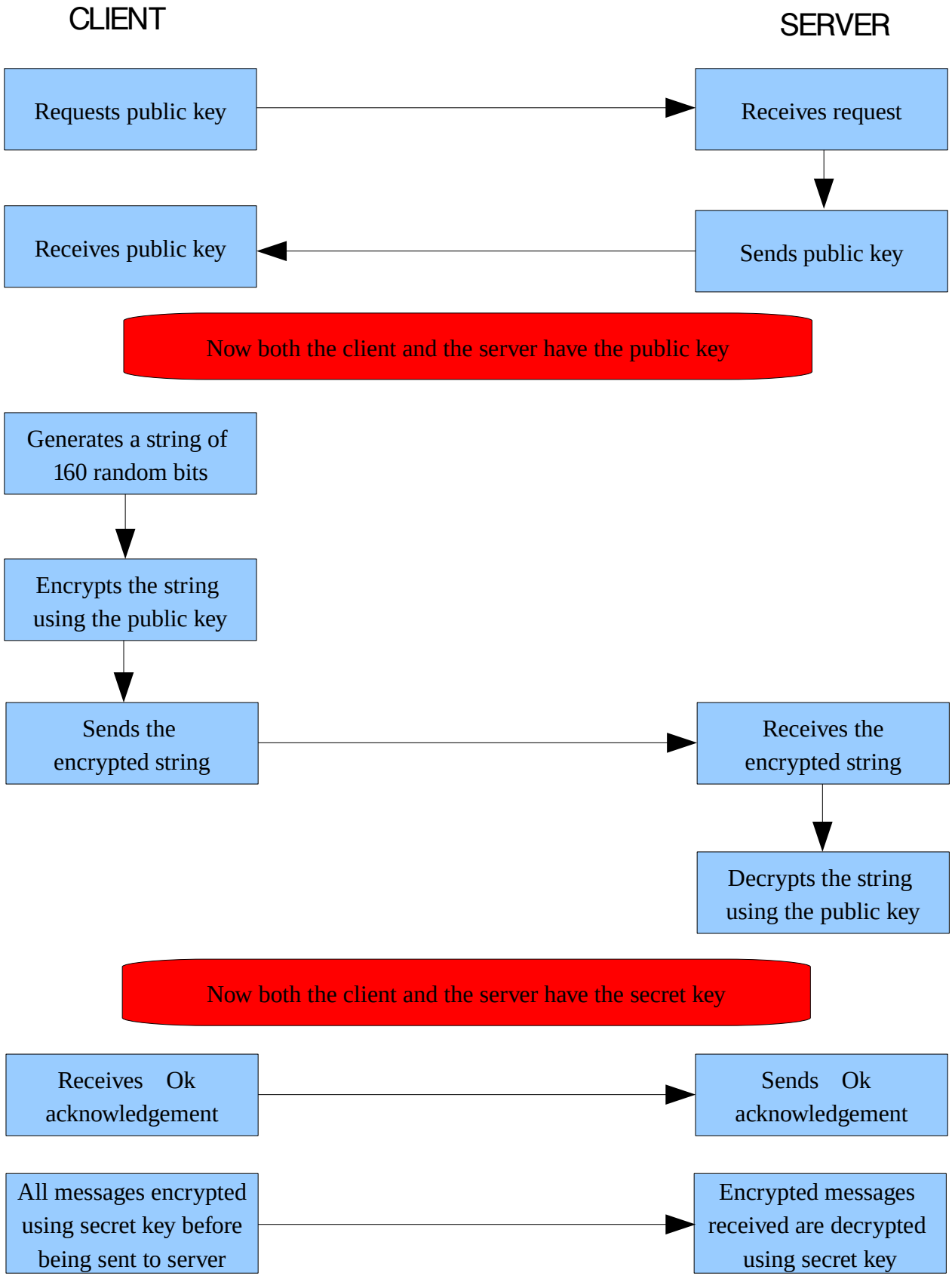
# Reliable Transmission Layer




- First of all a packet is received in the reliable transmission layer (sent from the socket layer) using the `rel_receive_packet` function
- The packet is checked to see if the correct packet has been received using its fragment number:
  - If the packet received is indeed the packet expected `(packet->header.fragment == required_frag)` then an acknowledgement is sent back to the socket layer
  - If the packet received is the same as the previous packet received `(packet->header.fragment == (required_frag)-1)` then no acknowledgement for that packet has been received by the socket layer, so the packet has been resent. An acknowledgement is sent to the socket layer.
  - If the packet received is neither the expected packet or the packet received previously, then it is resent.

- When the acknowledgement is sent to the socket layer a timer of 100ms starts:
  - If it is received before the time runs out `(if (socket_readable_timeout(socket, 100)))` then the acknowledgement is checked to see if the correct one has been received `((ack.header.fragment == required_frag) && (ack.header.is_data == 0))`:
  - If the ack received is correct, the socket layer sends the next packet to the reliable transmission layer and the process starts again.
  - If the ack received is not the one expected then it is resent.
  - If the acknowledgement is not received before the time runs out then the packet is resent and runs through this process again.

# Security Layer



- Firstly client sends a plain text string “get public key” to server `(frag_send_data(socket, get_pubkey, sizeof(get_pubkey), verbose, reliable))`
- The server sends back the public key `(frag_receive_data(socket, public_key, &length_pubkey, verbose, reliable))`
- The client generates a string of 160 random bits `(secret_key[j] = ((char)(rand()%(160))))`
- The client then encrypts the secret key (the string of 160 random bits) using the public key *(*`sec_pk_encrypt(secret_key, encrypted_pubkey, *public_key))`
- The client sends the encrypted secret key to the server *(*`frag_send_data(socket, encrypted_pubkey, ENCRYPTED_KEY_SIZE, verbose, reliable))`
- Once it has received the encrypted secret key, the server sends an acknowledgement to the client *(*`frag_receive_data(socket, &ok, &ok_length, verbose, reliable))`
- All future messages between server and client are encrypted and decrypted using the secret key:
  - An encrypted message is passed up from the fragmentation layer *(*`frag_receive_data(socket, data, &data_length, verbose, reliable))` which the server then decrypts using the secret key `(sec_symmetric_decrypt(data[0], data_length, pBuf, pLen, secret_key))`
  - The server encrypts a message using the secret key `(sec_symmetric_encrypt(data, len, data2, &data_length, secret_key))` and then passes it to the fragmentation layer `(frag_send_data(socket, data2[0], data_length, verbose, reliable))`

# Testing

To test my implementation of the client I used the functions time() and clock() to accurately time how long it takes to transfer files of different sizes and with reliable and secure transmission both off and on.

| Size of file | 293.2KB | | 41.0KB | |
|---|---|---|---|---|
| | Time(s) | Throughput (bits/s) | Time(ms) | Throughput (bits/s) |
| None | 0.3 | 7819 | 0.1 | 3280 |
| Reliable only | 35 | 149 | 5 | 66 |
| Secure only | 0.4 | 5864 | 0.2 | 1640 |
| Both | 36 | 65 | 5.1 | 64 |

It is easy to see that the bigger file took longer to receive. When neither reliable transmission nor security is on, the transfer is quickest. When only security is on, the transfer is still pretty quick – the only extra work being done is establishing the public and secret key and encrypting and decrypting the data. When only reliable transmission is on the file transfer takes much longer which is because reliable transmission means some packets are deliberately dropped, so the system waits until the timeout has been exceeded before resending them. This takes up a lot of time when this happens to many packets. When both security and reliable transmission is on the file transfer takes the longest as expected, since it has to do the most extra work.

The system survives packet drops since if a packet is dropped it is not received and because of the stop-and-wait algorithm in my programming, an acknowledgement of receipt of one packet must be received before the next packet is sent, and if the acknowledgement is not received (which will happen if the packet is dropped) then the packet is sent again.

# Security Services provided by the protocol in the security layer

There are three methods of ensuring the security of data: confidentiality, integrity and authenticity. There are two different types of data encryption: symmetric and asymmetric.

## Confidentiality

Since we are encrypting our data it is readable only by those who have access to the public and secret keys. Thus it is confidential (unless someone has gained unauthorised access to the public and secret keys, or just worked out what they are). To ensure the highest level of confidentiality and ensure the encryption is very unlikely to be broken the key size needs to be big enough. The smaller the key the easier it is to work out or guess what it is, but the bigger the key the more combinations of random bits it has, so it is much more difficult to crack – a key of length n bits is one of $2^n$ possible keys. As n increases the number of possible keys increases exponentially, which is very rapid growth, so the longer the key the greater the number of possible keys. According to Claude Shannon, for a message to be completely secure the size of the key needs to be at least the size of the message and the key can only be used once.

The security layer uses a secret key of length 160, so there are a possible $1.461501637 \times 10^{48}$ keys of the same length. This is secure enough for the purposes of the coursework but the key would have to be a lot longer if encrypting much more valuable data, for example in bank transactions.

## Integrity

Data integrity involves ensuring that data is correct and consistent at the beginning and at the end of the encryption/decryption process. Data integrity is lost by changing the data either accidentally (e.g. a transmission error) or deliberately (e.g. a hacker changing an account number in a bank transaction). MD5 is a hash function widely used to protect data integrity.

The security layer does not include any methods for ensuring data integrity. Whilst this should be improved it is not necessary since the data being encrypted is not valuable. However, in the encryption of valuable data it is essential to protect data integrity and the were my coursework to be used, it would definitely have to be improved.

## Authenticity

Authenticity is making sure the server and the client are who they say they are. It is ensuring that they both are confident that they are communicating with a trusted party. The security layer does not include any authentication procedures, which again should be improved but is not essential considering the data we are using is not valuable. Once way of ensuring data authenticity is to use the Challenge-Response Principle, where one party asks the other a question which they must answer correctly. A very simple example would be one party challenging the other for a password – the right password would ensure data authenticity. (However this is a little too simple – anybody could find out the password so this is not a reliable way to prove data authenticity. One way of making it more reliable is to use multiple passwords, for example.)

## Asymmetric Encryption

Two different keys are used to encrypt a message and decrypt it. A message is encrypted using a public key, which is available to anybody, and decrypted using a private key, which is unique to the user and they must keep it secret. Each user has their own public and private key and to send an encrypted message to someone for them to decrypt, it must have been encrypted using their public key.

## Symmetric Encryption

Both the sender and the receiver share a secret key which is used to encrypt and decrypt all messages.