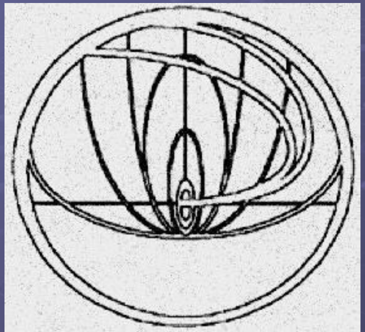
Systems and Devices 2 (Network)
Lec 3a: Transport Layer

Before we get started ...

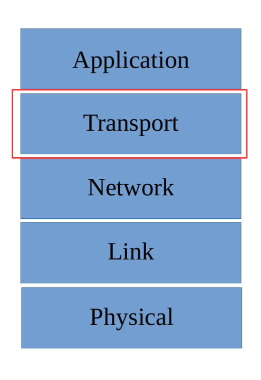
- How did your research into NTP go?
  - RFC 958 (5905): https://tools.ietf.org/html/rfc958
  - Fixed point representation: 32, 64 or 128 bit formats
  - Base date: 0 h 1 January 1900 UTC
    - Coordinated Universal Time (UTC)
  - Dooms day : 8 Feb 2036 (wrap around, reset to 0)
- From the top level view point we have all the protocols we need to get our network started, BUT, what is happening under the bonnet?
  - What are the UDP and TCP protocols we have identified in the transport layer?
  - How do we actually pass data between machines?
  - Why do we need two types of protocols?

Before we get started ...



- John Titor : in 2000 posts appeared on discussion boards claiming to be an American military time traveller from 2036, looking for a IBM 5100 computer, so that he could fix various legacy machines in the future, as it was not susceptible to this problem.

Internet protocol stack

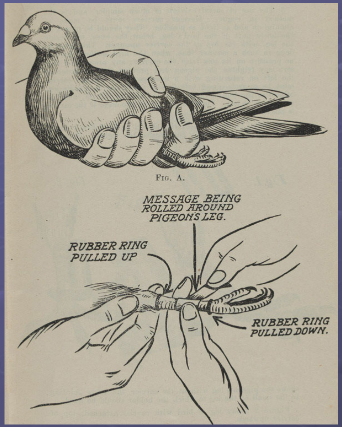


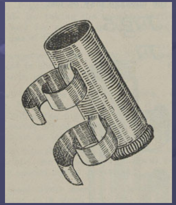
- Application layer
- Transport
  - Breaks messages down into **segments** that will be transferred. Also deals with error detection, congestion control and retransmission in the event of lost packets.
    - Transmission Control Protocol (TCP)
    - User Datagram Protocol (UDP)
- Network
- Link
- Physical

IPoAC

- To help explain these concepts we will consider one of the most important internet protocols ever developed : IPoAC
  - RFC 1149: https://tools.ietf.org/html/rfc1149
  - RFC 2549: https://tools.ietf.org/html/rfc2549
  - RFC 6214: https://tools.ietf.org/html/rfc6214
  - Note, typically updated in April. Especially high data rates in London, located around Trafalgar Square.

IPoAC





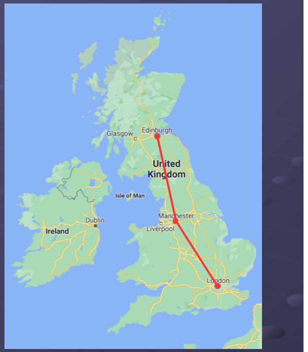
- IP over Avian Carriers
  - Message encapsulated in a metal canister attached to leg

IPoAC

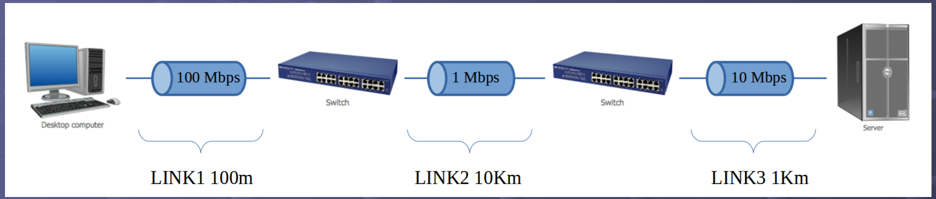
- Multiple packets in flight at the same time, these may be received out of order. Unfortunately packet lose may also occur :(

IPoAC

- Delays associated with “transmitting” packets from London to Edinburgh
  - Processing delay : time taken to load message into canister.
    - Also may need to encrypt message
  - Queuing delay : pigeons that have already arrived from different messages i.e. already in the hutch and waiting to be released.
  - Transmission delay : time taken to move pigeons out of the hutch and “throw” into the air, avoiding mid-air collisions. D = Len / Rate
  - Propagation delay : Time Of Flight (ToF)
- $D_{End-End} = N_{stages}(D_{proc} + D_{Que} + D_{Tran} + D_{Prop})$

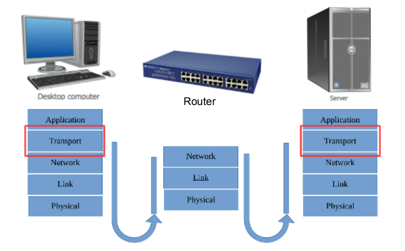


Quick Quizzz



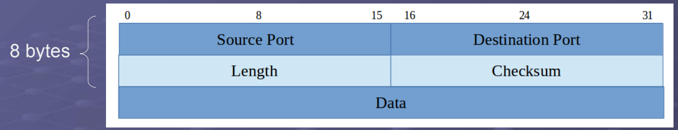
- If we assume that the propagation delay on each link is 2x10^8 m/sec i.e. max is speed of light, but slower in different materials, and that packet length is 1500 bits.
  - What is the transmission and propagation delays of each link
  - What is the end-to-end delay to transfer each packet?
    - You will need to make some assumptions / estimations.

Pause to consider ...



- The transport layer provides a logical communications path between client and server.
  - Not concerned about the network structure, or how data is transmitted e.g. pigeons :)
  - Application layer processes can use this end-to-end communications link to pass data (segments) i.e. a logical link between transport layers on “different” machines.
    - Routers / switches do not need to process the transport layer to route packets across the network i.e. a logical communication link between hosts is provided by the network layer.

UDP

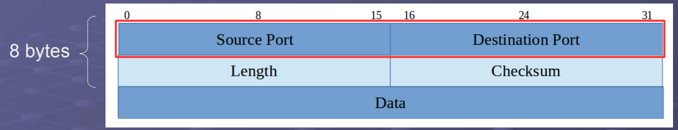


8 bytes

0 8 15	16 24 31
Source Port	Destination Port
Length	Checksum
Data	

- User Datagram Protocol
  - RFC 768 : https://tools.ietf.org/html/rfc768
    - In RFC UDP packets referred to as a datagrams, but this name also used to describe network layer, so to avoid confusion will refer to as segments :)
  - Created in 1983 as a simple, low overhead (small header) communications protocol.
    - Very simple Ports, Length and Checksum.
  - Defined as an unreliable, connectionless service
    - Best-effort delivery service, no handshaking / ACK, as soon as data is ready TX i.e. fire and forget.

UDP

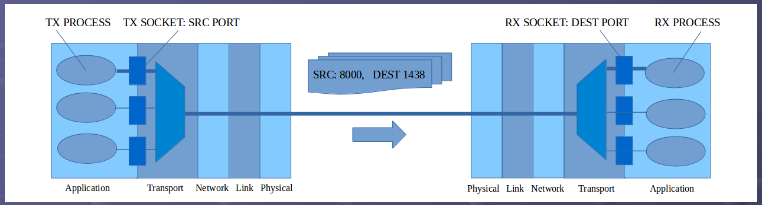


8 bytes

0 8 15	16 24 31
Source Port	Destination Port
Length	Checksum
Data	

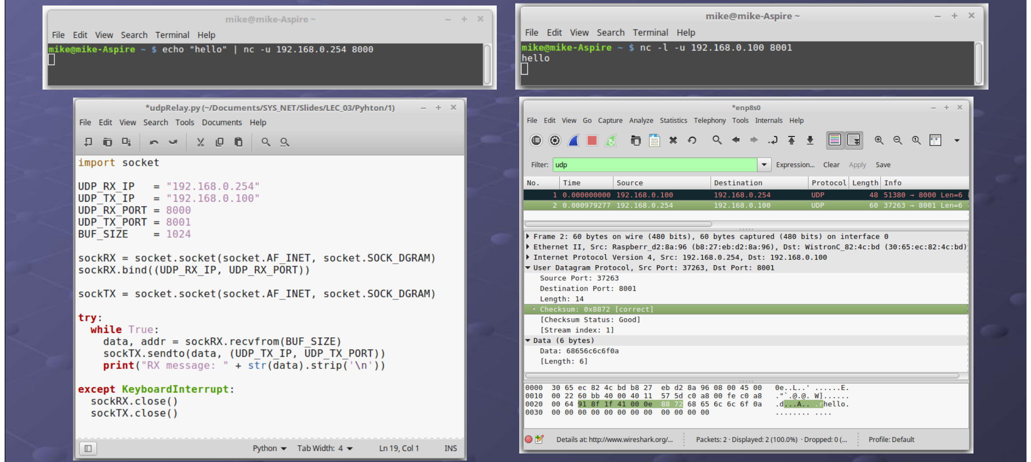
- User Datagram Protocol
  - RFC 768 : https://tools.ietf.org/html/rfc768
    - In RFC UDP packets referred to as a datagrams, but this name also used to describe network layer, so to avoid confusion will refer to as segments :)
  - Created in 1983 as a simple, low overhead (small header) communications protocol.
    - Very simple Ports, Length and Checksum.
  - Defined as an unreliable, connectionless service
    - Best-effort delivery service, no handshaking / ACK, as soon as data is ready TX i.e. fire and forget.

Pause to consider ...



- Q : hosts can have multiple network connections open at a time. How does it decide which process gets received data and how does it identify the process on a remote host to send data to?
  - Port numbers : 16bit port numbers 0 - 65335
    - IANA: https://bit.ly/35mKNR9
    - Categorised as system, registered and ephemeral ports
  - Network layer performs host-to-host data delivery. Process-to-process data delivery is transport-layer **multiplexing** and **demultiplexing** via socket port numbers.

Demo


mike@mike-Aspire ~
File Edit View Search Terminal Help
*udpRelay.py (~/Documents/SYS_NET/Slides/LEC_03/Pyhton/1)
File Edit View Search Tools Documents Help
Python
Tab Width: 4
Ln 19, Col 1
INS
*enp8s0
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Expression...
Clear
Apply
Save

```
mike@mike-Aspire ~ $ echo "hello" | nc -u 192.168.0.254 8000
```

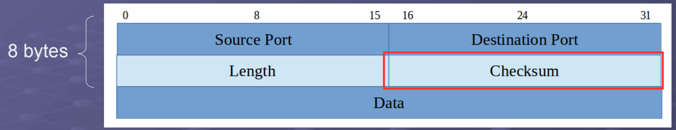
```
mike@mike-Aspire ~ $ nc -l -u 192.168.0.100 8001
hello
```

```
import socket

UDP_RX_IP   = "192.168.0.254"
UDP_TX_IP   = "192.168.0.100"
UDP_RX_PORT = 8000
UDP_TX_PORT = 8001
BUF_SIZE    = 1024

sockRX = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sockRX.bind((UDP_RX_IP, UDP_RX_PORT))

sockTX = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

try:
  while True:
    data, addr = sockRX.recvfrom(BUF_SIZE)
    sockTX.sendto(data, (UDP_TX_IP, UDP_TX_PORT))
    print("RX message: " + str(data).strip('\n'))

except KeyboardInterrupt:
  sockRX.close()
  sockTX.close()
```

```
Filter: udp

No.  Time         Source         Destination    Protocol Length Info
   1 0.000000000  192.168.0.100  192.168.0.254  UDP          48 51380 → 8000 Len=6
   2 0.000979277  192.168.0.254  192.168.0.100  UDP          60 37263 → 8001 Len=6

▸ Frame 2: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
▸ Ethernet II, Src: Raspberr_d2:8a:96 (b8:27:eb:d2:8a:96), Dst: WistronC_82:4c:bd (30:65:ec:82:4c:bd)
▸ Internet Protocol Version 4, Src: 192.168.0.254, Dst: 192.168.0.100
▾ User Datagram Protocol, Src Port: 37263, Dst Port: 8001
    Source Port: 37263
    Destination Port: 8001
    Length: 14
  ▸ Checksum: 0x8872 [correct]
    [Checksum Status: Good]
    [Stream index: 1]
▾ Data (6 bytes)
    Data: 68656c6c6f0a
    [Length: 6]

0000  30 65 ec 82 4c bd b8 27  eb d2 8a 96 08 00 45 00   0e..L..' ......E.
0010  00 22 60 bb 40 00 40 11  57 5d c0 a8 00 fe c0 a8   ."`.@.@. W]......
0020  00 64 91 8f 1f 41 00 0e  88 72 68 65 6c 6c 6f 0a   .d...A.. .rhello.
0030  00 00 00 00 00 00 00 00  00 00 00 00               ........ ....

Details at: http://www.wireshark.org/...   Packets: 2 · Displayed: 2 (100.0%) · Dropped: 0 (...   Profile: Default
```

- Simple python based UDP relay server

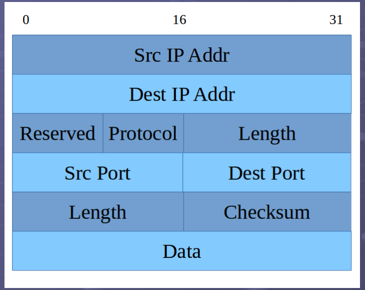
UDP



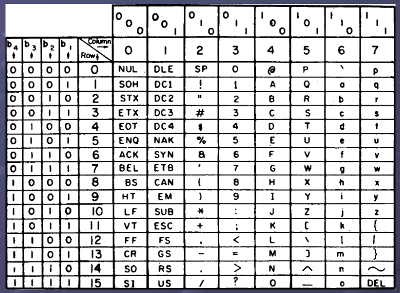
Source Port	Destination Port
Length	Checksum
Data	

- User Datagram Protocol
  - RFC 768 : https://tools.ietf.org/html/rfc768
    - In RFC UDP packets referred to as a datagrams, but this name also used to describe network layer, so to avoid confusion will refer to as segments :)
  - Created in 1983 as a simple, low overhead (small header) communications protocol.
    - Very simple Ports, Length and Checksum.
  - Defined as an unreliable, connectionless service
    - Best-effort delivery service, no handshaking / ACK, as soon as data is ready TX i.e. fire and forget.

UDP checksum



Src IP Addr		
Dest IP Addr		
Reserved	Protocol	Length
Src Port		Dest Port
Length		Checksum
Data		



					0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b4 ↓	b3 ↓	b2 ↓	b1 ↓	Column → Row ↓	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	|
1	1	0	1	13	CR	GS	−	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

- To calculate the UDP checksum we use a pseudo header
  - IP addresses included to ensure that the segment received (packet) is for this host and not a corrupted segment.
  - Checksum calculation must be efficient to avoid increasing processing delays. Note, checksum offloading in HW.
    - https://tools.ietf.org/html/rfc1071

UDP checksum

Accumulation	Bit inversion	Check
1111101011110101	00000010 = 0x02	0x0294
+0000011110011110	10010100 = 0x94	+0xFD6B
10000001010010011	= 0x0294	0xFFFF
0000001010010011	11111101 = 0xFD	
+0000000000000001	01101011 = 0x6B	
0000001010010100	= 0xFD6B	

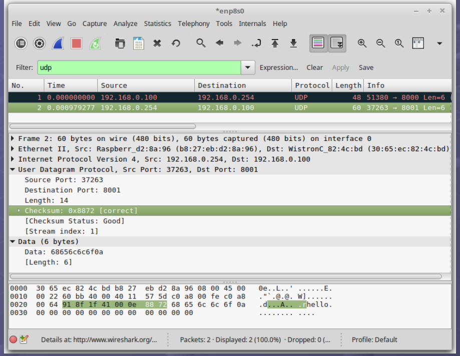
- TX: UDP checksum generated through repeated addition
  - 32bit blocks broken down into 16bit values and accumulated
  - Any overflows wrapped round (bit 16) and added back into the 16bit accumulation, repeated if second overflow
  - Final result inverted and stored in UDP header.

UDP checksum

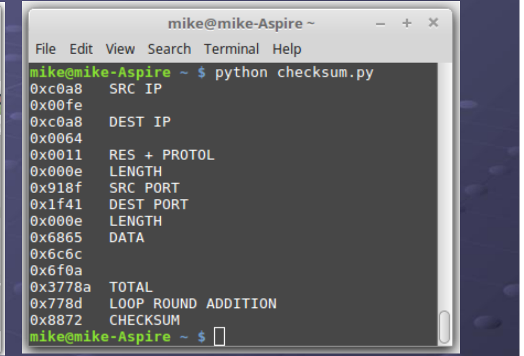
Accumulation	Bit inversion	Check
1111101011110101	00000010 = 0x02	0x0294
+0000011110011110	10010100 = 0x94	+0xFD6B
10000001010010011	= 0x0294	0xFFFF
0000001010010011	11111101 = 0xFD	
+0000000000000001	01101011 = 0x6B	
0000001010010100	= 0xFD6B	

- RX: UDP checksum regenerated (removing checksum)
  - Receiving host regenerates checksum and adds received checksum to it. Should generate the result 0xFFFF. If not an error has been detected.

Demo


*enp8s0
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: udp
Expression... Clear Apply Save

```
No.  Time         Source         Destination    Protocol Length Info
   1 0.000000000  192.168.0.100  192.168.0.254  UDP          48 51380 → 8000 Len=6
   2 0.000979277  192.168.0.254  192.168.0.100  UDP          60 37263 → 8001 Len=6

▸ Frame 2: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
▸ Ethernet II, Src: Raspberr_d2:8a:96 (b8:27:eb:d2:8a:96), Dst: WistronC_82:4c:bd (30:65:ec:82:4c:bd)
▸ Internet Protocol Version 4, Src: 192.168.0.254, Dst: 192.168.0.100
▾ User Datagram Protocol, Src Port: 37263, Dst Port: 8001
    Source Port: 37263
    Destination Port: 8001
    Length: 14
  ▸ Checksum: 0x8872 [correct]
    [Checksum Status: Good]
    [Stream index: 1]
▾ Data (6 bytes)
    Data: 68656c6c6f0a
    [Length: 6]

0000  30 65 ec 82 4c bd b8 27  eb d2 8a 96 08 00 45 00   0e..L..' ......E.
0010  00 22 60 bb 40 00 40 11  57 5d c0 a8 00 fe c0 a8   ."`.@.@. W].......
0020  00 64 91 8f 1f 41 00 0e  88 72 68 65 6c 6c 6f 0a   .d...A.. .rhello.
0030  00 00 00 00 00 00 00 00  00 00 00 00               ........ ....

Details at: http://www.wireshark.org/...   Packets: 2 · Displayed: 2 (100.0%) · Dropped: 0 (...   Profile: Default
```



```
mike@mike-Aspire ~ $ python checksum.py
0xc0a8   SRC IP
0x00fe
0xc0a8   DEST IP
0x0064
0x0011   RES + PROTOL
0x000e   LENGTH
0x918f   SRC PORT
0x1f41   DEST PORT
0x000e   LENGTH
0x6865   DATA
0x6c6c
0x6f0a
0x3778a  TOTAL
0x778d   LOOP ROUND ADDITION
0x8872   CHECKSUM
mike@mike-Aspire ~ $
```

- UDP checksum generation
  - To simplify the process, performed using a simple python script.

Quick Quizzz

```
 01101101
 00101011
+11001100
_________

_________
```

- Generate the 8bit checksum for the above data.
- If the received checksum is 0x65 has an error occurred?
- Is UDP a connection or connectionless based protocol?
- NTP uses UDP port 123, what happens if a data segment is dropped in the network i.e. what happens if no time data is returned to the client? Does this matter?

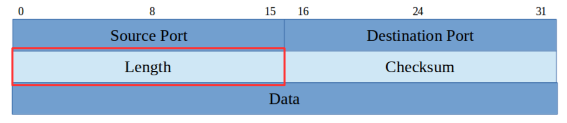
UDP

- Advantages of UDP:
  - Low overheads: small header (8 bytes), no connection establishment delays. Application code can transmit segment as soon as it is ready.
  - Speed: low overheads and data transmitted at full data rate i.e. no congestion control, does not have to wait for ACK.
  - Minimise server loads: do not need to record connection state, send buffers etc. Therefore, server can support more clients.
  - UDP is ideal for real-time applications (less delay) that can tolerant some packet loss.

UDP

- Disadvantages of UDP
  - No error control, corrupt segments dropped i.e. fail silent. Applications layer must be able to tolerant these scenarios.
    - Can push re-transmit functionality up into the application layer.
  - No congestion control i.e. hosts transmitting large amounts of data can swamp a network i.e. block connections from other machines :(.
  - No confirmation / acknowledgement that the receiving host has received the transmitted data i.e. unreliable, segments could get lost.

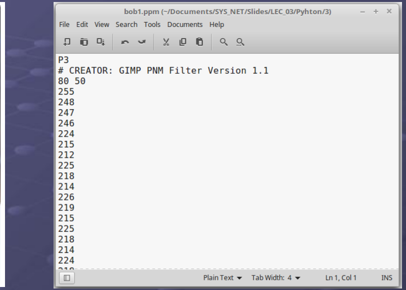
UDP



0 8 15	16 24 31
Source Port	Destination Port
Length	Checksum
Data	

- Q : what is the biggest possible UDP segment size?
  - RFC 791 states that the max “safe” packet size is 576B
    - “a reasonable sized data block”
    - UDP segment <= 576B : all host must be accept packets of this size i.e. guaranteed to be deliverable, but not guaranteed to be delivered.
    - UDP segment > 576B : are allowed to be dropped by a router / hosts. Limitations in the router / switch hardware
  - Max possible UDP payload dependent on lower layers, but for now we will say approximately 64K - (IP + UDP) headers.
    - UDP packet may be fragmented by lower layers i.e. split up into smaller chunks If any fragment lost the entire packet is dropped.
    - MTU: Maximum Transmission Unit, for IP frames this is ~1500 bytes.

Programming Task

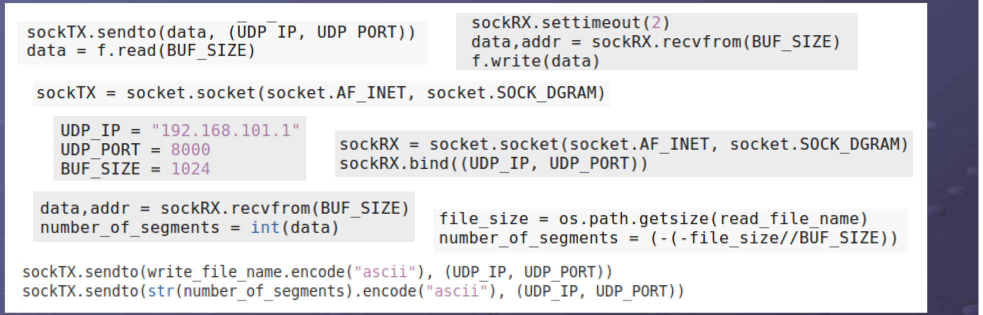


bob1.ppm (~/Documents/SYS_NET/Slides/LEC_03/Pyhton/3)
File Edit View Search Tools Documents Help
Plain Text
Tab Width: 4
Ln 1, Col 1
INS

```
P3
# CREATOR: GIMP PNM Filter Version 1.1
80 50
255
248
247
246
224
215
212
225
218
214
226
219
215
225
218
214
224
```

- Write two python programs ie. TX and RX, to transfer a ppm image file of Bob (text file) between PC and Pi.
  - Hint: the file may be larger than the “max” UDP size, break the file down into chunks using the file.read(BUF) method.
- Q : what port should you use? Could you use port 123?

Hints



```
sockTX.sendto(data, (UDP_IP, UDP_PORT))
data = f.read(BUF_SIZE)
```

```
sockRX.settimeout(2)
data,addr = sockRX.recvfrom(BUF_SIZE)
f.write(data)
```

```
sockTX = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

```
UDP_IP = "192.168.101.1"
UDP_PORT = 8000
BUF_SIZE = 1024
```

```
sockRX = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sockRX.bind((UDP_IP, UDP_PORT))
```

```
data,addr = sockRX.recvfrom(BUF_SIZE)
number_of_segments = int(data)
```

```
file_size = os.path.getsize(read_file_name)
number_of_segments = (-(-file_size//BUF_SIZE))
```

```
sockTX.sendto(write_file_name.encode("ascii"), (UDP_IP, UDP_PORT))
sockTX.sendto(str(number_of_segments).encode("ascii"), (UDP_IP, UDP_PORT))
```

- What your software needs to do:
  - Send the image file name and the number of segments that will be transmitted i.e. MTU = 1024B.
  - Send the image data as multiple MTU sized segments.
  - Received segments are stored to a file using TX file name.

Summary

- UDP
  - Fast : low overhead (8bytes) communications protocol
    - Note, remember that the UDP header and data will be passed to lower layers in the stack, each adding additional headers e.g. IP header can vary from 20 to 60 bytes.
  - Connectionless : no time wasted setting up a connection between the two machines i.e. as soon as data is ready transmit onto the network and hope the RX host is ready :)
  - Efficient : do not need to reserve memory in server to store connection status or send data buffers i.e. reduces server memory loading.
  - Unreliable : a bit misleading, best effort delivery service.
    - Note, don’t confuse this term with faulty, rather that segment delivery is not guaranteed.

Summary

- UDP is used to implement:
  - DNS, NTP, DHCP, BOOTP, TFTP, SNMP, RTP, RTSP, …
  - Steam (game client) ports 27000–27030-ish.
- To overcome the limitations of UDP we need a different protocol:
  - TCP : Transmission Control Protocol
  - For applications were we need to guarantee that data is delivered e.g. email.
  - For next week have a look into this protocol, its a LOT more complex, and do finish off the UDP file transfer coding quizzz.