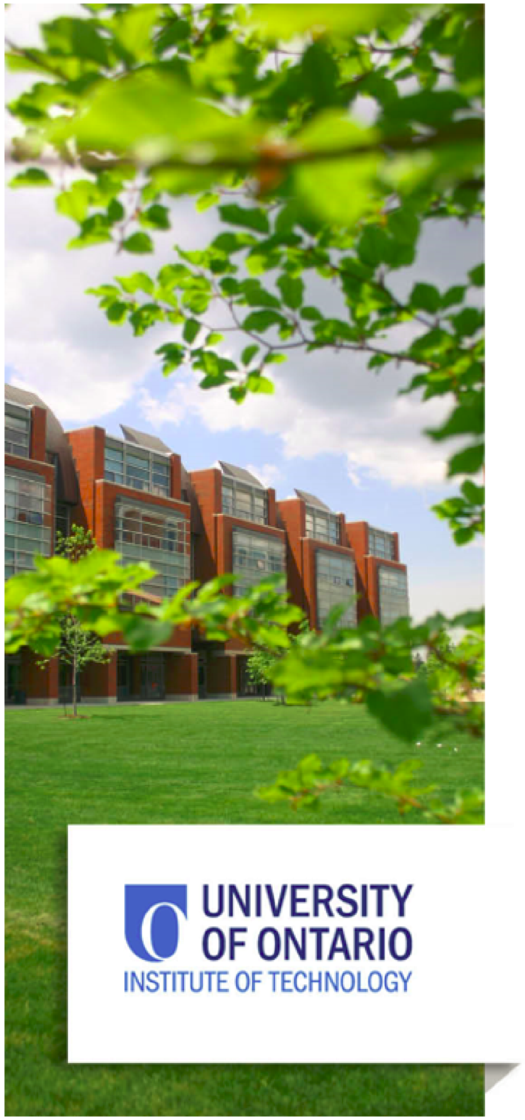


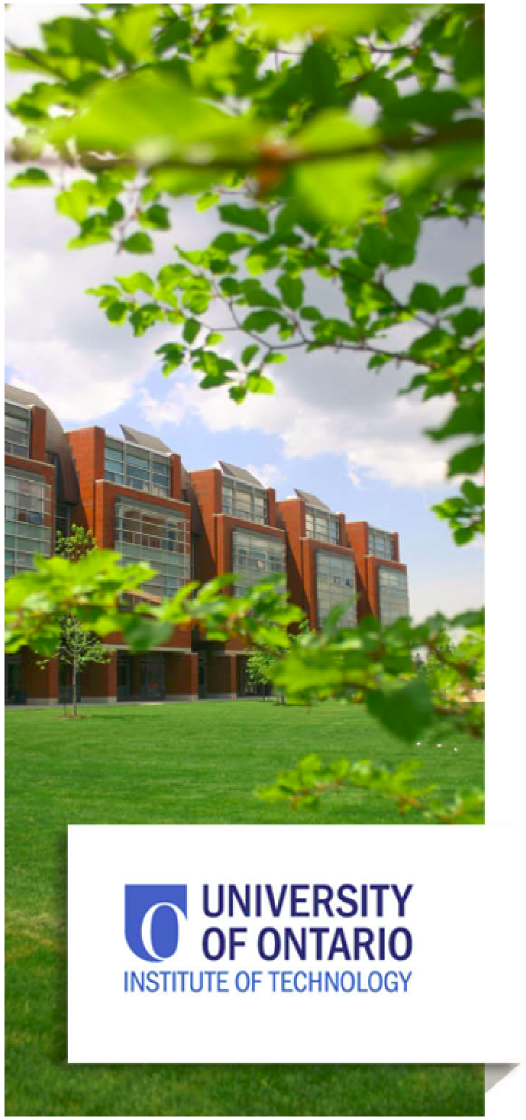
TCP, UDP, Sockets, Open Data

Writing Network Programs

Randy J. Fortier
randy.fortier@uoit.ca
@randy_fortier

Outline

- TCP
    - Ports
    - Handshake
    - Windowing
    - Checksum
    - Packet Format
- UDP
    - Packet Format
- Socket Programming
    - Stream Sockets
    - Datagram Sockets

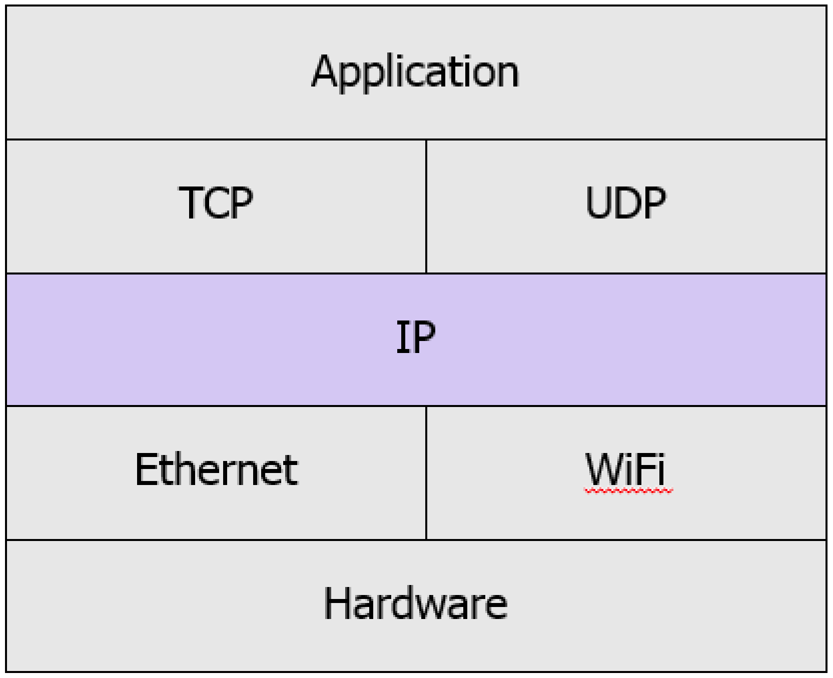


TCP, UDP, Sockets, Open Data

Internet Protocol

Internet Protocol

- Sending packets to their destination

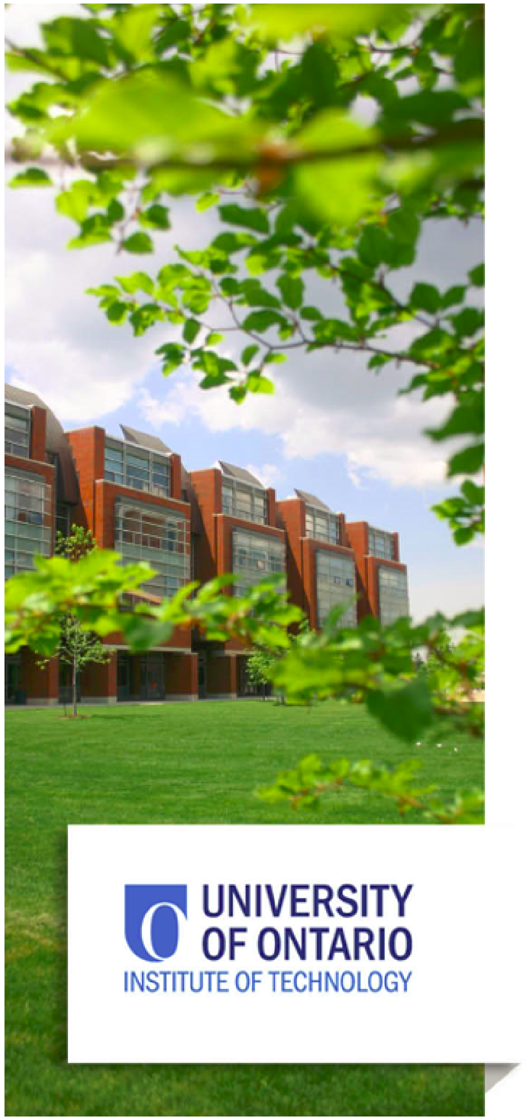


IP

- Addressing
- Routing

IP Addresses

- IPv4
  - 4 bytes (32 bits)
  - Written in decimal, separated by dots
  - e.g. 192.197.54.136
- IPv6
  - 8 double bytes (128 bits)
  - Written in hexadecimal, separated by colons
  - e.g. FFE80:0000:0000:0000:0000:0000:AC1E:43FE
  - Short form: FFE80:AC1E:43FE

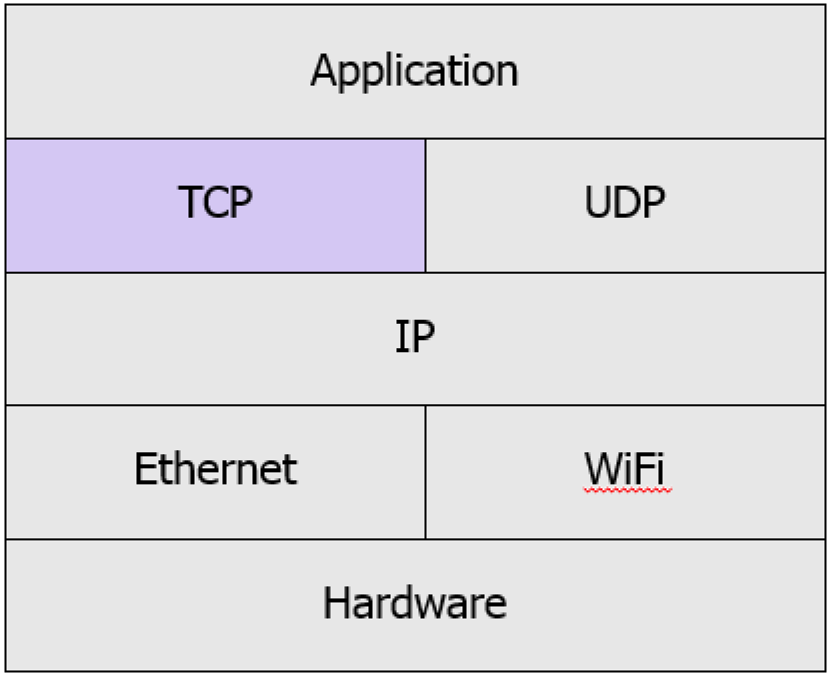


TCP, UDP, Sockets, Open Data

Transport Control Protocol

Transport Control Protocol

- Adding reliability to packet delivery



TCP

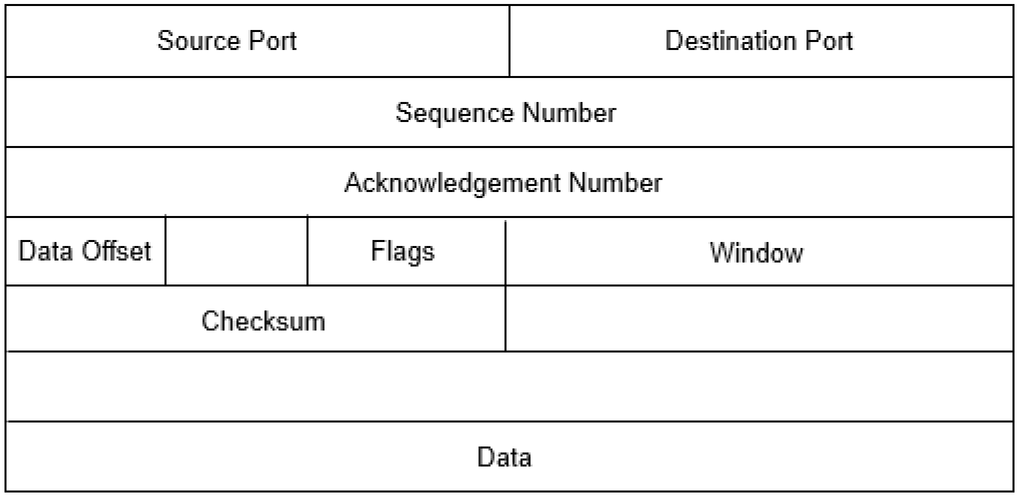
- Reliable
  - Streams
  - Connections
  - Sequence numbers
  - Acknowledgements (ACKs)
  - Error checking (Checksums)

Streams

- In most networks, data is sent in finite quanta, called packets
- Output streams are implemented with packets in a similar way to saving data to files via blocks:
  - Data to be sent is collected in a buffer
  - When there is enough data to fill an entire packet is collected, a packet is transmitted
- Input streams are implemented with packets in a similar way to reading data from files via blocks:
  - When you want to read a single character/byte, the buffer is checked
  - As packets arrive, their data is added to the buffer

TCP Packet Format

- Every TCP packet has a header (meta-data):
  1. Flags
  2. Sequence Number
  3. Source and Destination Port Number

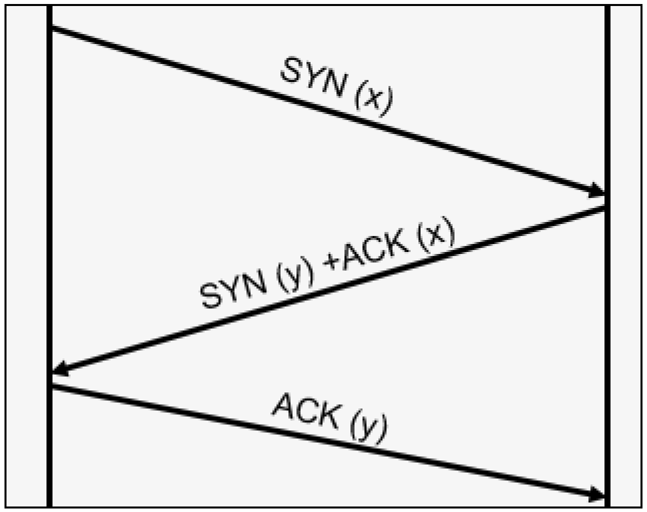


Flags

- Single-bit binary values:
  1. SYN - attempting to establish a connection
  2. FIN - attempting to tear down a connection
  3. ACK - this packet contains an acknowledgement
  4. RST - attempting to reset the connection

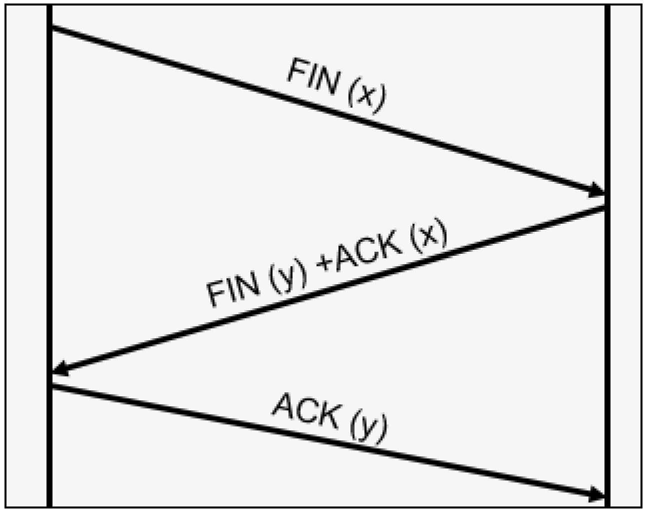
Connection Handshake

- Before communication can take place, a connection is established
- This is a three-step process:
  1. Client sends a connection request (SYN)
  2. Server acknowledges connection request (SYN+ACK)
  3. Client acknowledges receipt of server's response (ACK)



Disconnection Handshake

- To disconnect, another three-way handshake occurs:
  1. Client sends a connection finished request (FIN)
  2. Server acknowledges connection finished request (FIN+ACK)
  3. Client acknowledges receipt of server's response (ACK)

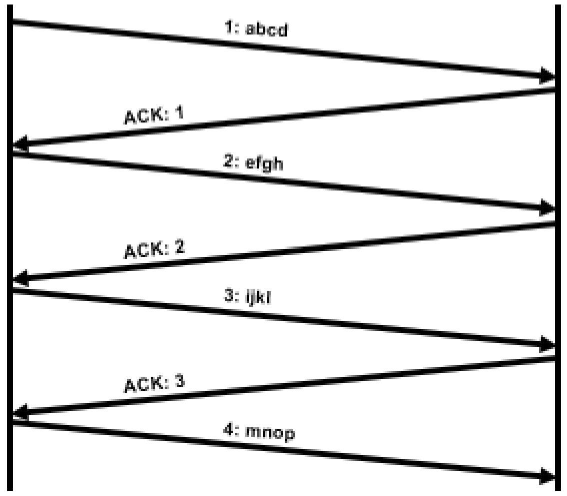


Sequence Numbers

- Each packet has a randomized, hard-to-predict sequence number
- This sequence number is used when acknowledging a packet

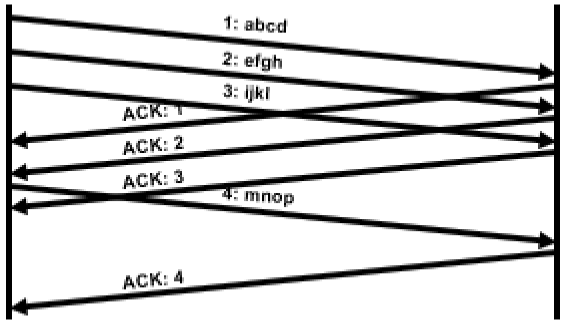
Acknowledgements

- Each packet is acknowledged (by its sequence number) by sending a message back to the sender
- Without any improvements:



Acknowledgements

- Improvements:
  - Piggybacked acknowledgements
    - Acknowledgements are sent in messages that were already going to be sent back in response
  - Sliding window
    - Up to N unacknowledged packets can be out at once:

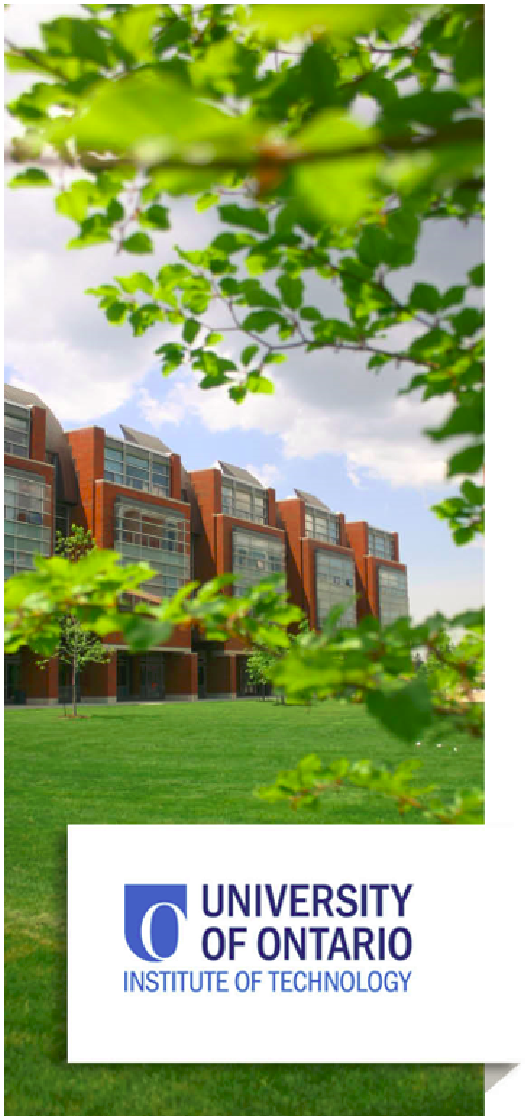


Port Numbers

- A destination port number serves a similar purpose as an IP address
    - IP address
        - Which computer on the network should receive this packet?
    - Destination port
        - Which application on that computer should receive this packet?
- A source port number is to give the receiving computer a way to contact the source application with its next message

Well-Known Port Numbers

- There are many standard port numbers:
  - 22: SSH
  - 25: Sending E-Mail
  - 53: DNS
  - 80: Unencrypted HTTP
  - 110/143: Receiving E-Mail
  - 443: Encrypted HTTP (HTTPS)

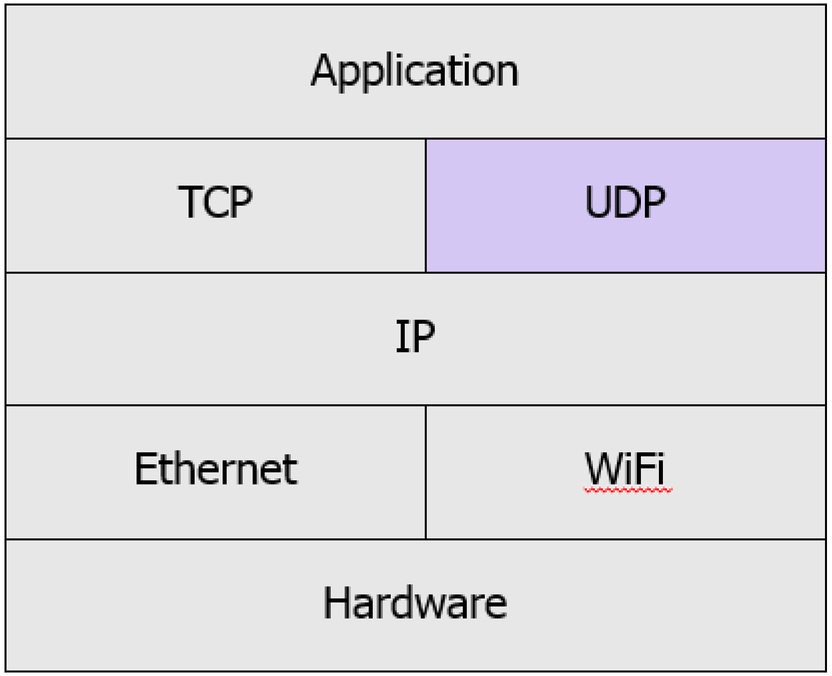


TCP, UDP, Sockets, Open Data

User Datagram Protocol

User Datagram Protocol

- Transport without the reliability baggage



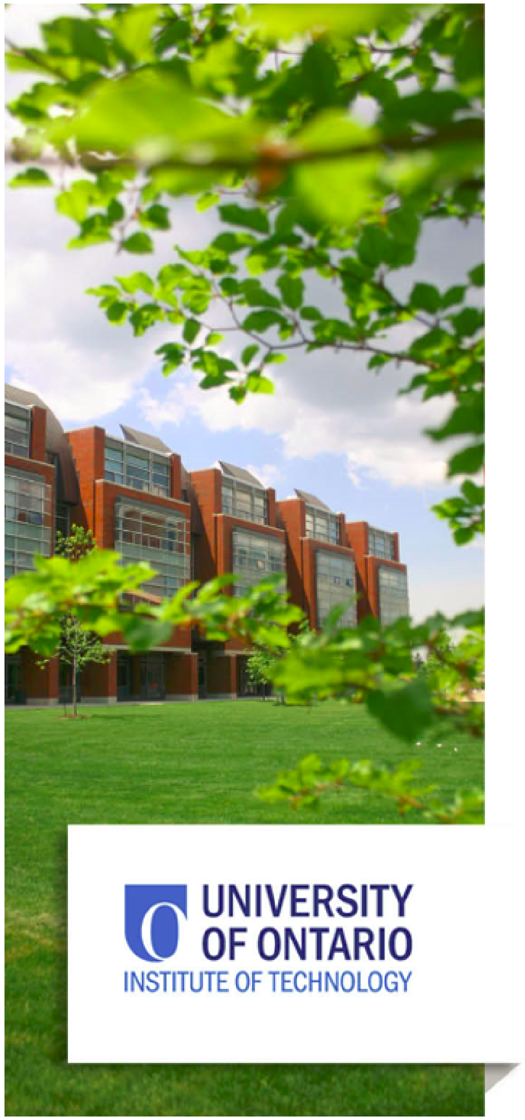
User Datagram Protocol

- Pros:
  - No acknowledgements
  - No connection set up/tear down
  - Supports multicast/broadcast
- Cons:
  - Lost packets are not identified and resent
  - Packets may arrive out of order
  - Does not implement streams

UDP Packet Format

- UDP packets have no flags, acknowledgements, sequence numbers:

Source Port	Destination Port
Length	Checksum
Data	



TCP, UDP, Sockets, Open Data

TCP Socket Programming

TCP Sockets in Java

- Listens for incoming connections (server, passive open):

```
ServerSocket serverSocket = new ServerSocket(8080);
while (true) {
    Socket clientSocket = serverSocket.accept();
    ... input and output goes here ...
}
```

TCP Sockets in Java

- Connects to the server (client, active open):

```
Socket socket = new Socket("myhost.com", 8080);
... input and output goes here ...
```

TCP Sockets in Java

- Send output to a socket:

```
Socket socket = new Socket("myhost.com", 8080);
PrintWriter out = new PrintWriter(socket.getOutputStream());
String request = "GET /index.html HTTP/1.0\r\nHost:myhost.com\r\n\r\
out.print(request);
out.flush();
```

TCP Sockets in Java

- Get input from a socket:

```
Socket clientSocket = serverSocket.accept();
InputStream inStream = clientSocket.getInputStream();
InputStreamReader reader = new InputStreamReader(inStream);
BufferedReader in = new BufferedReader(reader);
String line = null;
while ((line = in.readLine()) != null) {
    // do something with 'line'
}
```

TCP Sockets in Java

- Disconnect from a socket:

```
Socket clientSocket = serverSocket.accept();
...
clientSocket.close();
```

TCP Sockets in Python

- Listens for incoming connections (server, passive open):

```
serverSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
serverSocket.bind((socket.gethostname(), 8080))
serverSocket.listen(5) # 5 requests queued
while True:
    (clientSocket, address) = serverSocket.accept()
    ... input and output goes here ...
```

TCP Sockets in Python

- Connects to the server (client, active open):

```
socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
socket.connect(("myhost.com", 8080))
... input and output goes here ...
```

TCP Sockets in Python

- Send output to a socket:

```
socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
socket.connect(("myhost.com", 8080))
request = "GET /index.html HTTP/1.0\r\nHost:myhost.com\r\n\r\n"
socket.send(request)
```

TCP Sockets in Python

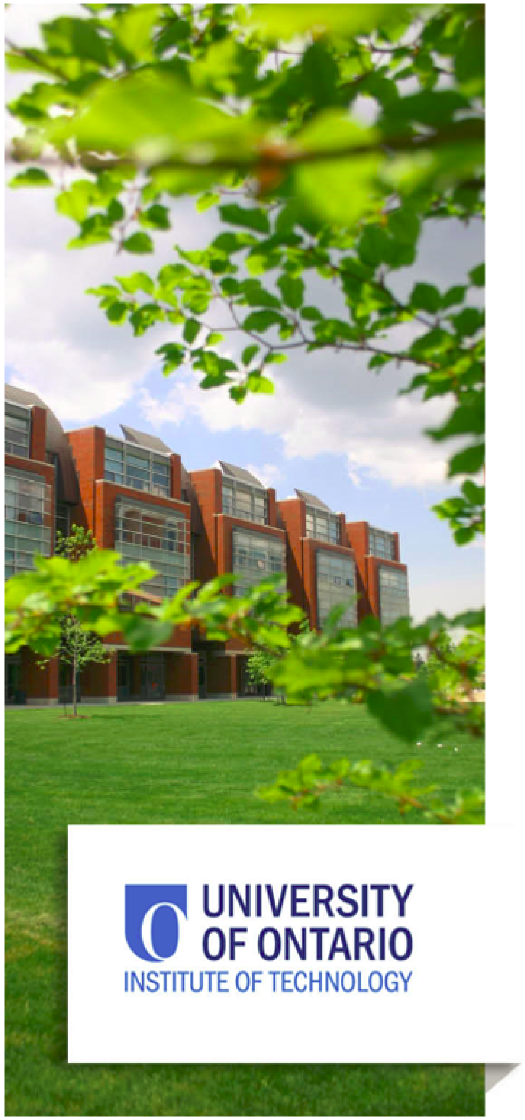
- Get input from a socket:

```
(clientSocket, address) = serverSocket.accept()
receivedBytes = -1
while receivedBytes != 0:
    line = clientSocket.recv(2048)
    # do something with 'line'
```

TCP Sockets in Python

- Disconnect from a socket:

```
clientSocket = serverSocket.accept()
...
clientSocket.close()
```



TCP, UDP, Sockets, Open Data

UDP Socket Programming

UDP Sockets in Java

- Sending packets (user datagrams):

```
DatagramSocket socket  = new DatagramSocket(16789);
String msg = "hello";
String IP = "192.197.54.136";
// The following can also do DNS lookup
InetAddress address = InetAddress.getByName(IP);
DatagramPacket outputPacket = new DatagramPacket(msg.getBytes(),
                                                  msg.length(),
                                                  address, 12465);
socket.send(outputPacket);
```

UDP Sockets in Java

- Receiving packets (user datagrams):

```
DatagramSocket socket  = new DatagramSocket(16789);
byte[] data = byte[256];
DatagramPacket inputPacket = new DatagramPacket(data, 256);
socket.receive(inputPacket);
```

UNIVERSITY OF ONTARIO INSTITUTE OF TECHNOLOGY

TCP, UDP, Sockets, Open Data

Network Programming Architectures

Network Programming Architectures

- Client/server
- Peer to peer
- Hybrid

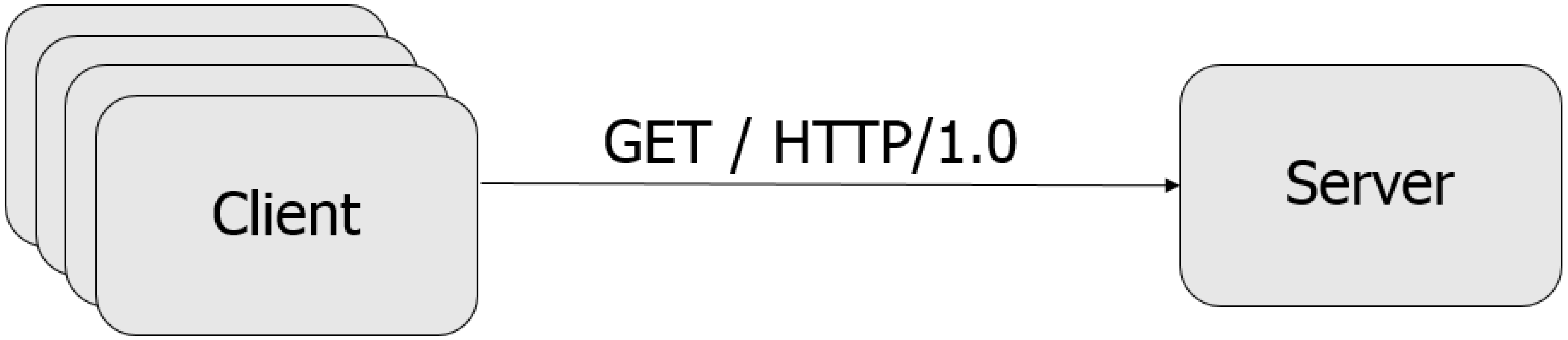
Client/server

- One server is accessed by a number of (identical?) clients
- The server is passively listening

Client
Server

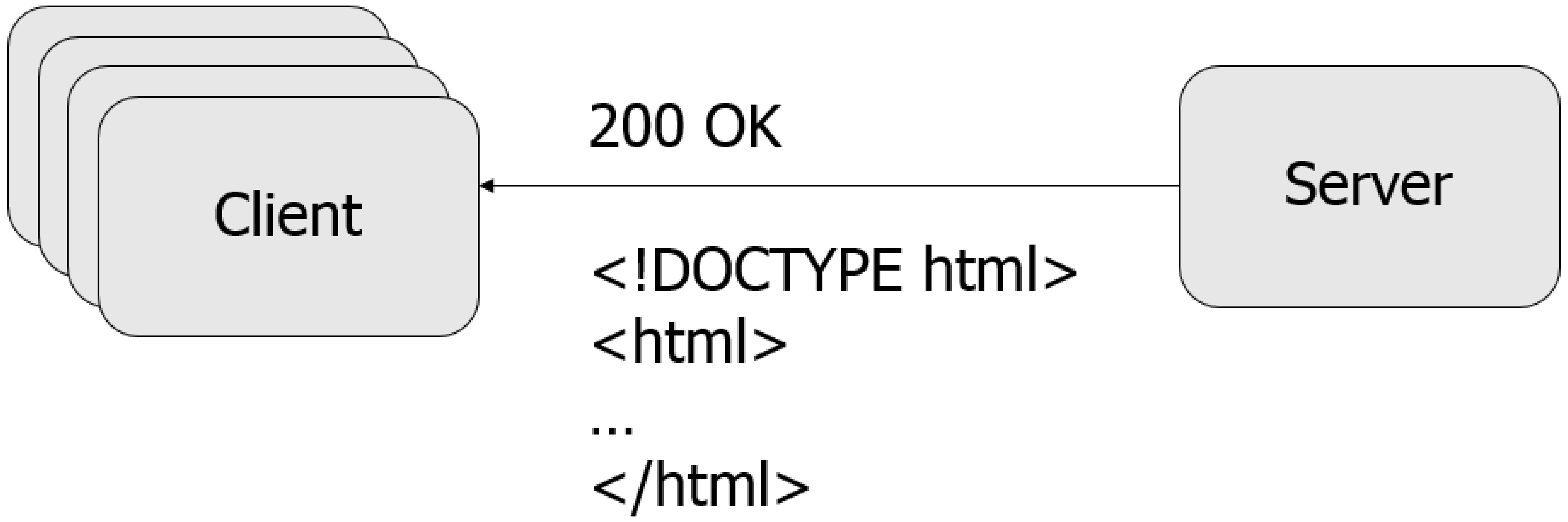
Client/server

- The client actively initiates the connection
- The client needs to know the IP address/port of the server
- This initial message is often called a request



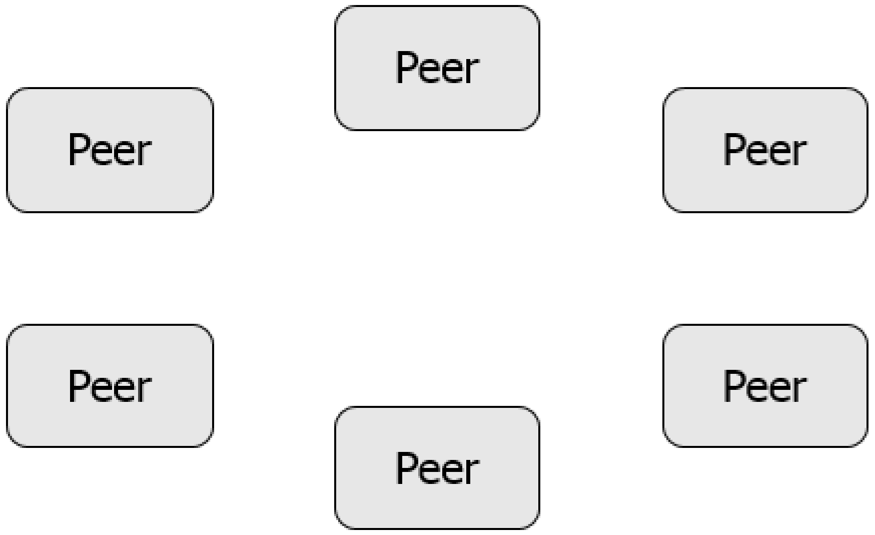
Client/server

- The server provides a response to the request
- The IP address/port from the request are used for the response



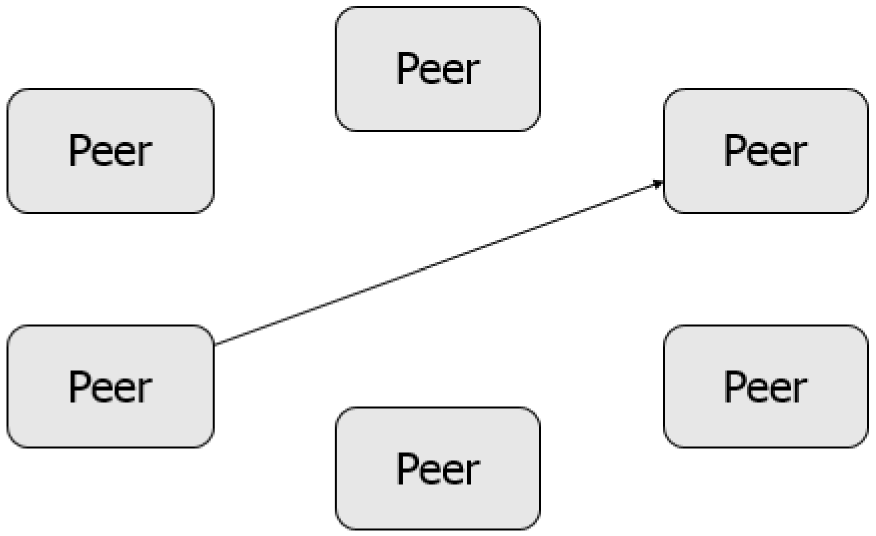
Peer to peer

- All (identical?) peers have the same purpose



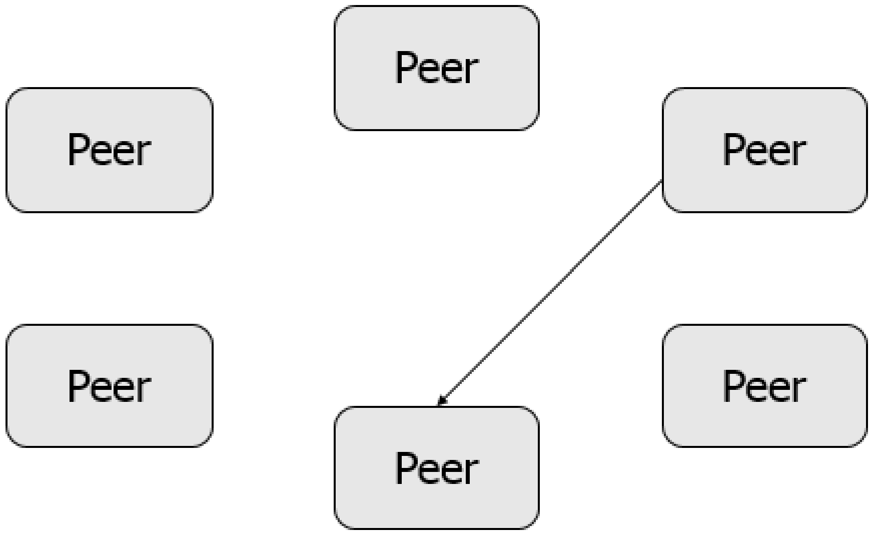
Peer to peer

- Any peer can send messages to any other



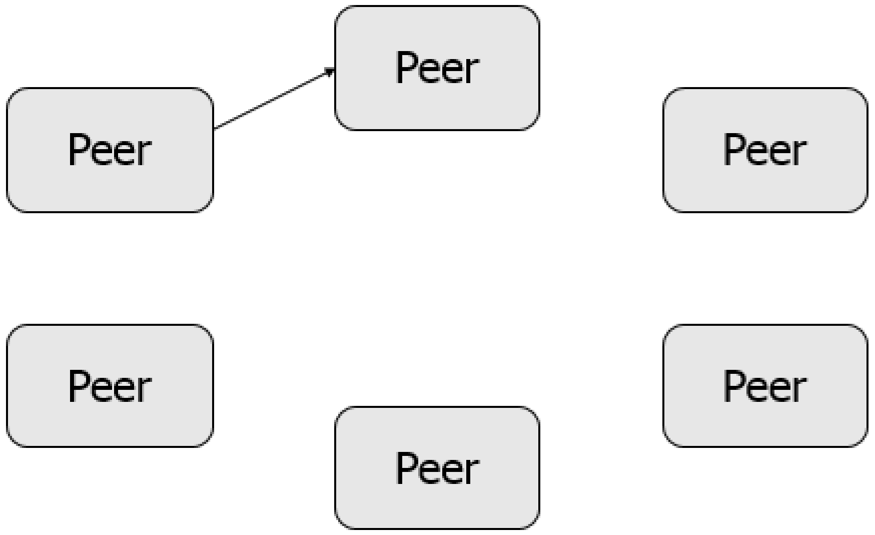
Peer to peer

- Any peer can send messages to any other



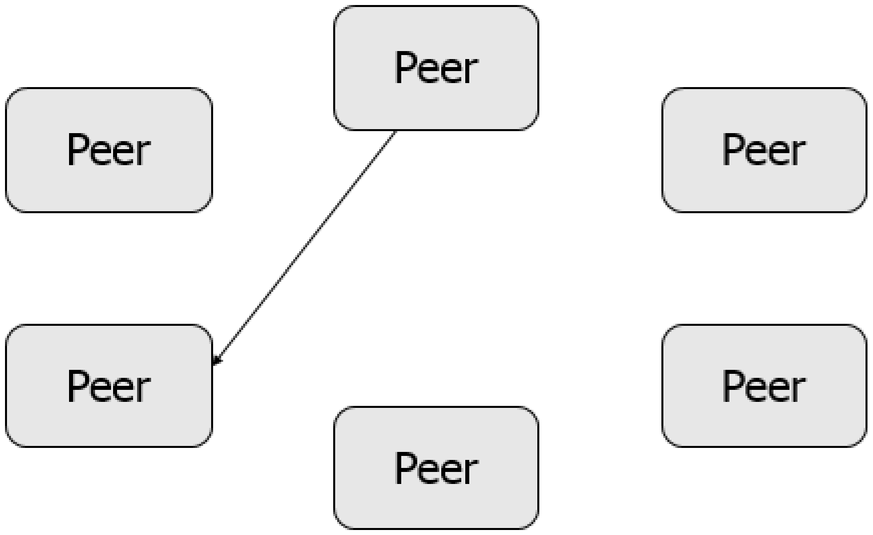
Peer to peer

- Any peer can send messages to any other



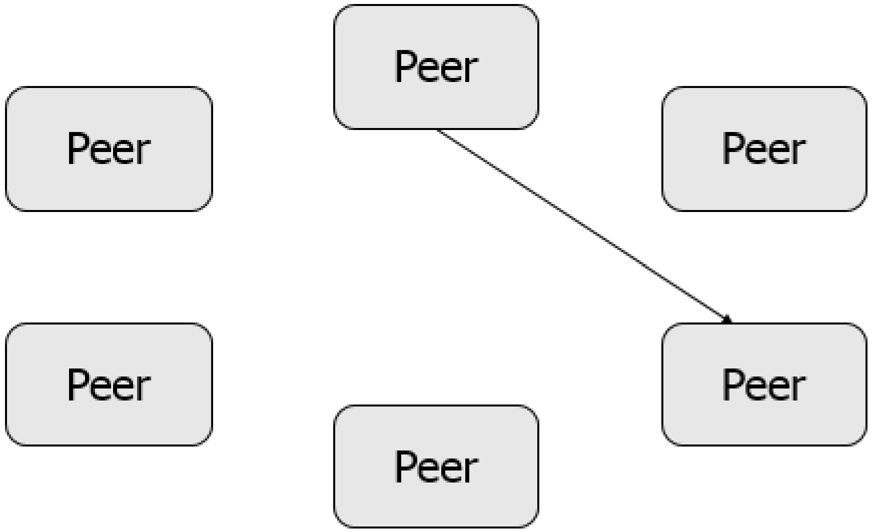
Peer to peer

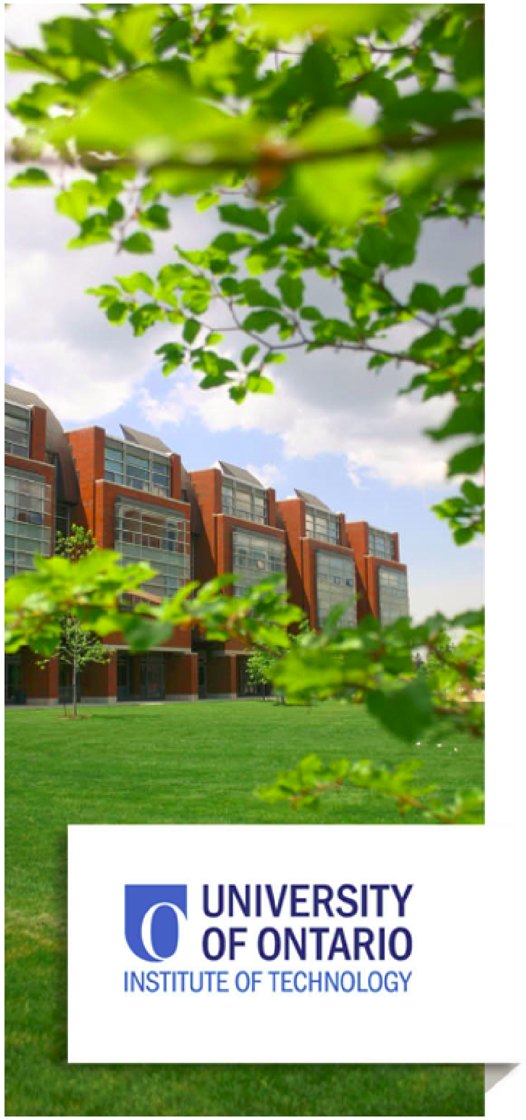
- Any peer can send messages to any other



Peer to peer

- Any peer can send messages to any other





TCP, UDP, Sockets, Open Data

Hypertext Transfer Protocol

HTTP

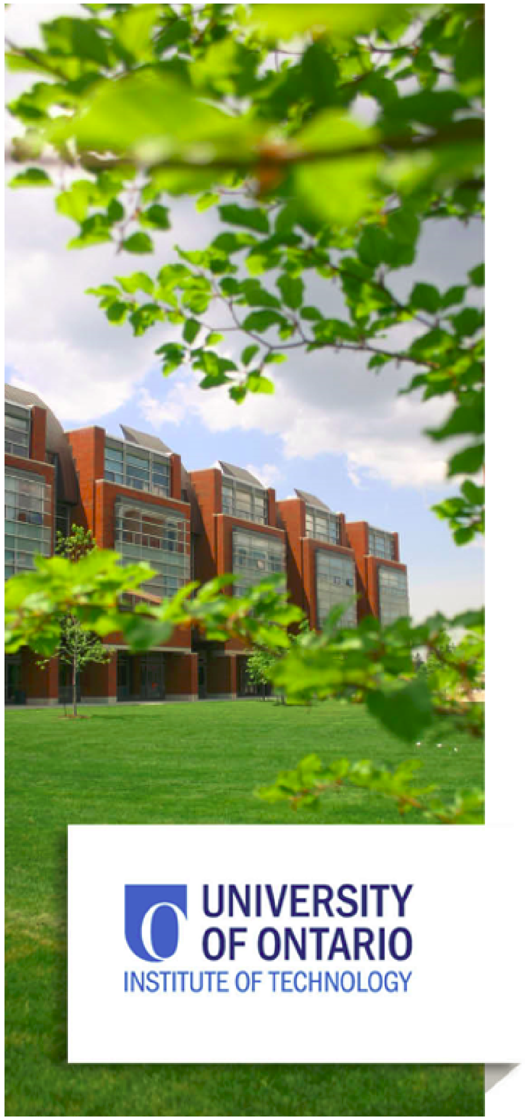
- Client/server protocol for uploading/downloading files
- An HTTP client (browser) issues a request:

```
GET / HTTP/1.1\r\n
Host: stackoverflow.com\r\n
Connection: keep-alive\r\n
Accept: text/html\r\n
Upgrade-Insecure-Requests: 1\r\n
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64)\r\n
Accept-Encoding: gzip, deflate\r\n
Accept-Language: en-US,en;q=0.8\r\n
Cookie: ...
```

HTTP

- Client/server protocol for uploading/downloading files
- The HTTP server (web server) issues a response:

```
HTTP/1.1 200 OK\r\n
Date: Thu, 14 Jan 2016 02:11:33 GMT\r\n
Content-Type: text/html; charset=utf-8\r\n
Content-Length: 40269\r\n
Connection: keep-alive\r\n
Cache-Control: public, max-age=31\r\n
Content-Encoding: gzip\r\n
Expires: Thu, 14 Jan 2016 02:12:05 GMT\r\n
Last-Modified: Thu, 14 Jan 2016 02:11:05 GMT\r\n
```

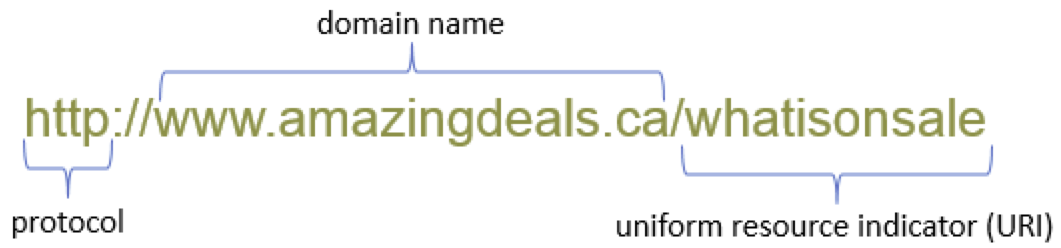


TCP, UDP, Sockets, Open Data

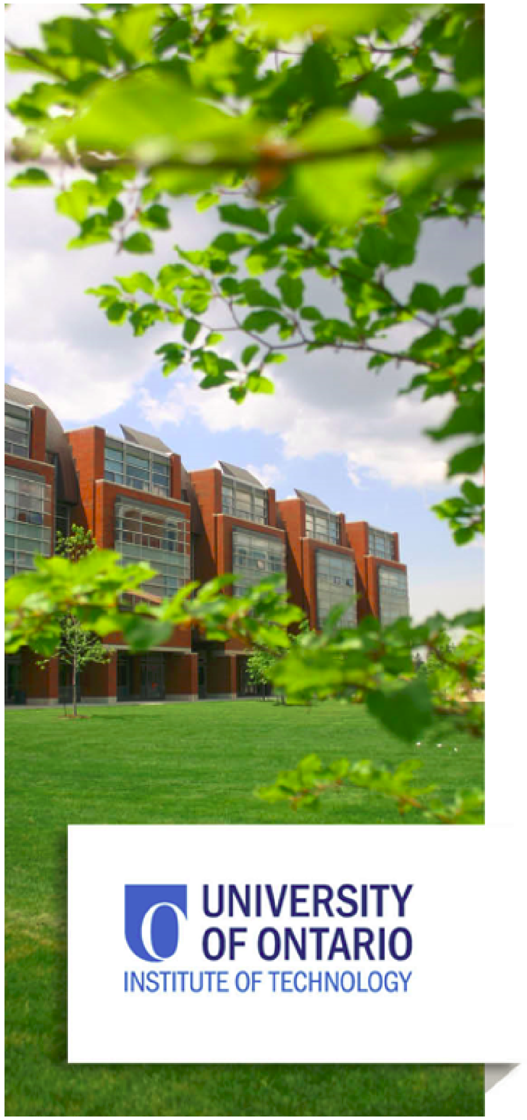
URL and URL Connections

URL

- Represents a URL
- Provides an easy way to connect to web servers


domain name
http://www.amazingdeals.ca/whatisonsale
protocol
uniform resource indicator (URI)

```
URL url = new URL("http://" + hostName + ":" + portNumber + "/index.
URLConnection conn = url.openConnection();
conn.setDoOutput(false);
conn.setDoInput(true);
InputStream is = conn.getInputStream();
...
```



TCP, UDP, Sockets, Open Data

Open Data

Open Data

- CSV
- JSON
- XML

CSV

- Comma-separated values
- e.g. http://open.canada.ca/data/en/dataset/ffe1ad5f-49c4-4d03-8b8e-25919d4481af

```
Ref_Date,GEO,VAR,SEX,AGE,STATS,CHAR,Vector,Coordinate,Value
2009,Canada,Body Mass Index (BMI) - Center for Disease Control (CDC)
2011,Canada,Body Mass Index (BMI) - Center for Disease Control (CDC)
2013,Canada,Body Mass Index (BMI) - Center for Disease Control (CDC)
...
```

JSON

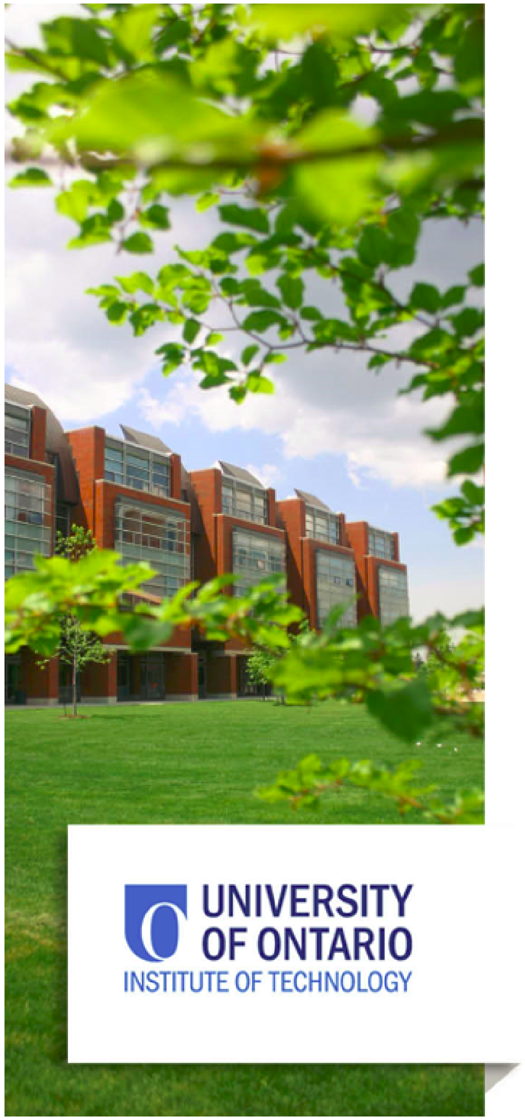
- JavaScript Object Notation
- e.g. http://api.coindesk.com/v1/bpi/currentprice/CAD.json

```
{"bpi":{
        "USD":{
               "code":"USD",
               "rate":"421.3680",
               "description":"United States Dollar",
               "rate_float":421.368},
        "CAD":{
               "code":"CAD",
               "rate":"553.1846",
               "description":"Canadian Dollar",
               "rate_float":553.1846}
        }
}
```

XML

- eXtensible Markup Language
- e.g. http://www.bikesharetoronto.com/data/stations/bikeStations.xml

```
<stations lastUpdate="1452772137445" version="2.0">
  <station>
   <id>1</id>
   <name>Jarvis St / Carlton St</name>
   <terminalName>7055</terminalName>
   <lastCommWithServer>1452771534819</lastCommWithServer>
   <lat>43.66207</lat>
   <long>-79.37617</long>
   <installed>true</installed>
   <locked>false</locked>
   ...
  </station>
</stations>
```



TCP, UDP, Sockets, Open Data

Processing Data

CSV

```
String csvData = ...;
String[] rows = csvData.split("\n");
for (int i = 0; i < rows.length; i++) {
    String[] cells = rows.split(",");
    ...
}
```

JSON

- This code uses the Simple JSON library
- Maven: http://mvnrepository.com/artifact/com.googlecode.json-simple/json-simple

```
BufferedReader jsonSource = ...;
JSONParser parser = new JSONParser();
JSONObject obj = (JSONObject)parser.parse(jsonSource);

/*{"bpi":{
        ...
        "CAD":{
                "code":"CAD",
                "rate":"553.1846",
                "description":"Canadian Dollar",
                "rate_float":553.1846}
        }
}*/
JSONObject bpi = (JSONObject)obj.get("bpi");
JSONObject cad = (JSONObject)bpi.get("CAD");
double rate = (Double)cad.get("rate_float");
```

XML

```
InputStream inStream = conn.getInputStream();
DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstanc
DocumentBuilder docBuilder = dbFactory.newDocumentBuilder();
Document document = docBuilder.parse(inStream);
document.getDocumentElement().normalize();

NodeList itemNodes = document.getElementsByTagName("stations");

for (int i = 0; i < itemNodes.getLength(); i++) {
   Node itemNode = itemNodes.item(i);
   if (itemNode.getNodeType() == Node.ELEMENT_NODE) {
      Element itemElement = (Element)itemNode;
      String name = getTagValue("name", itemElement);
      ...
   }
}
```

Wrap-Up

- In this section we learned about:
  - Internet Protocol (IP)
  - TCP and UDP
  - Socket Programming
  - HTTP and URLs
  - Open Data