

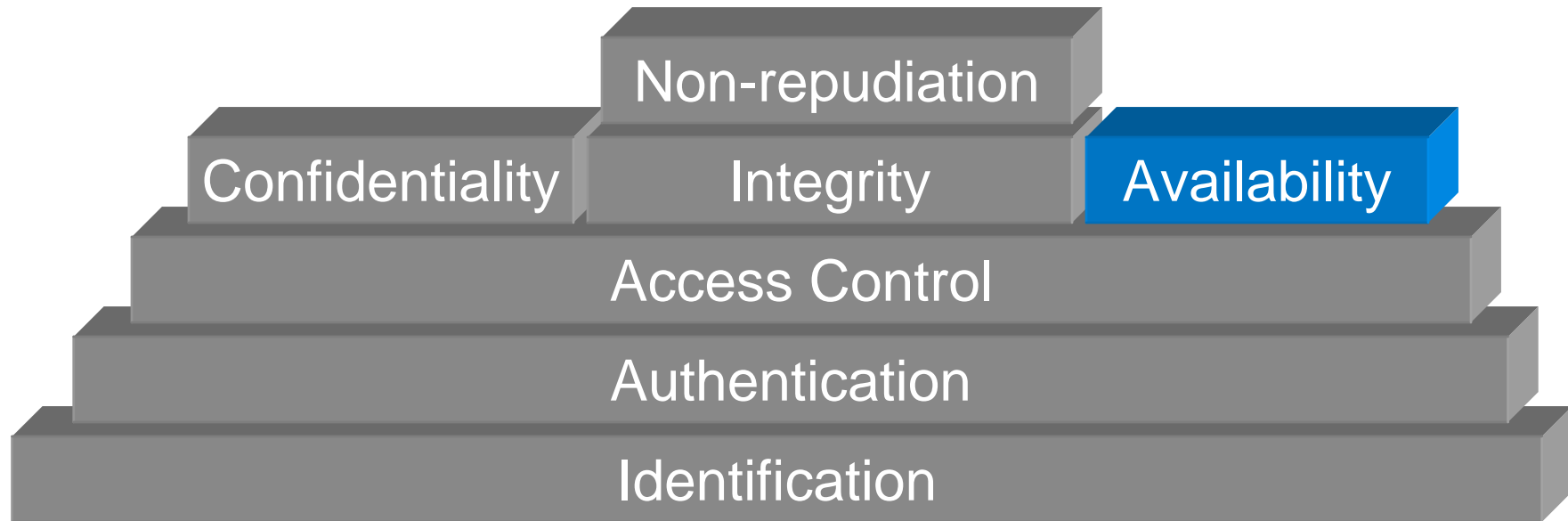
NIS/CpE 691A

Information System Security

**Stevens Institute of Technology
Spring 2006**

Class 6 – 2/23/06

One Structured Way of Viewing Security



Security Services

Hacking Through the Ages



Classes of Availability Attacks

- Consumption of a limited or non-renewable resource
 - Network connectivity
 - Using your own resources against you
 - Bandwidth usage
 - Other resources
- Destruction or modification of configuration information
- Physical destruction or modification of hardware

Classes of Availability Attacks

- **Consumption of a limited or non-renewable resource**
 - **Network connectivity**
 - **Using your own resources against you**
 - **Bandwidth usage**
 - **Other resources**
- Destruction or modification of configuration information
- Physical destruction or modification of hardware

UNIX Process Limits

```
#!/bin/sh  
exec $0 &  
sleep 60
```

- Consume all available processes on system – other users as well as system processes may be blocked
- Change “sleep” to another command and you might consume all the CPU cycles, as well.

UNIX File System Limits

```
!/bin/sh  
exec $0 &  
echo 'more stuff' > files$$
```

```
!/bin/sh  
cat bigfile >> biggerfile  
mv biggerfile bigfile  
exec $0
```

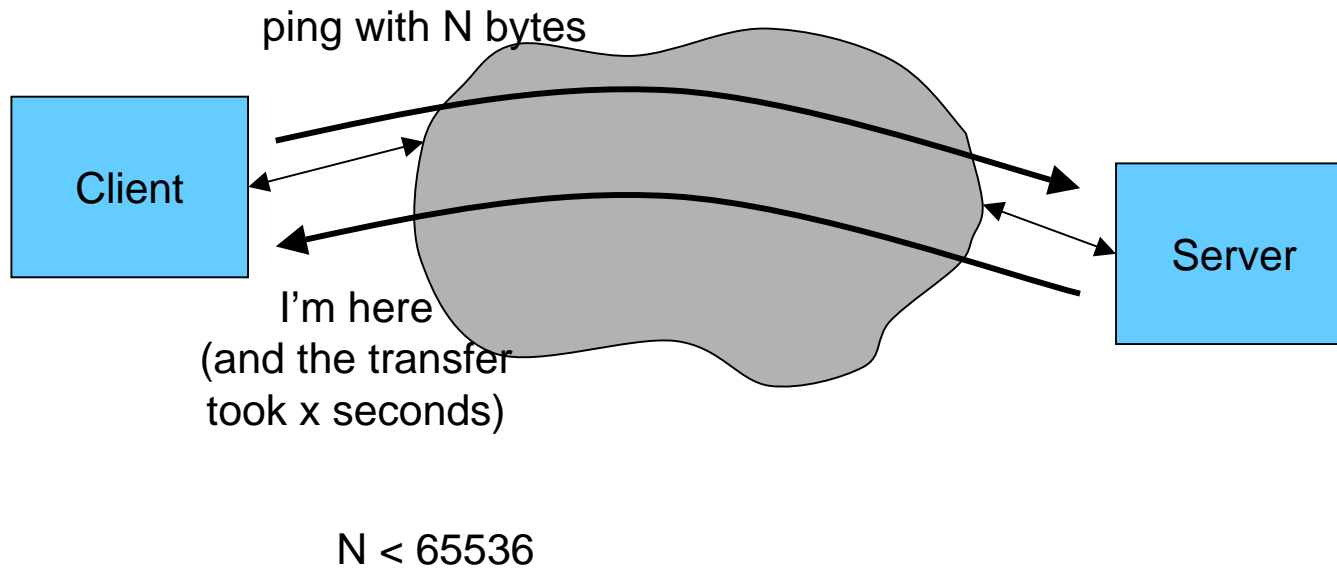
- Consume all available file system handles on system (i-nodes on a UNIX system)
- If you don't want to be limited by i-nodes first, you might fill the hard disk with the second shell

UNIX File System Limits

```
!/bin/sh  
exec $0 &  
echo `hello` | mail `cat /etc/passwd | cut -f1`
```

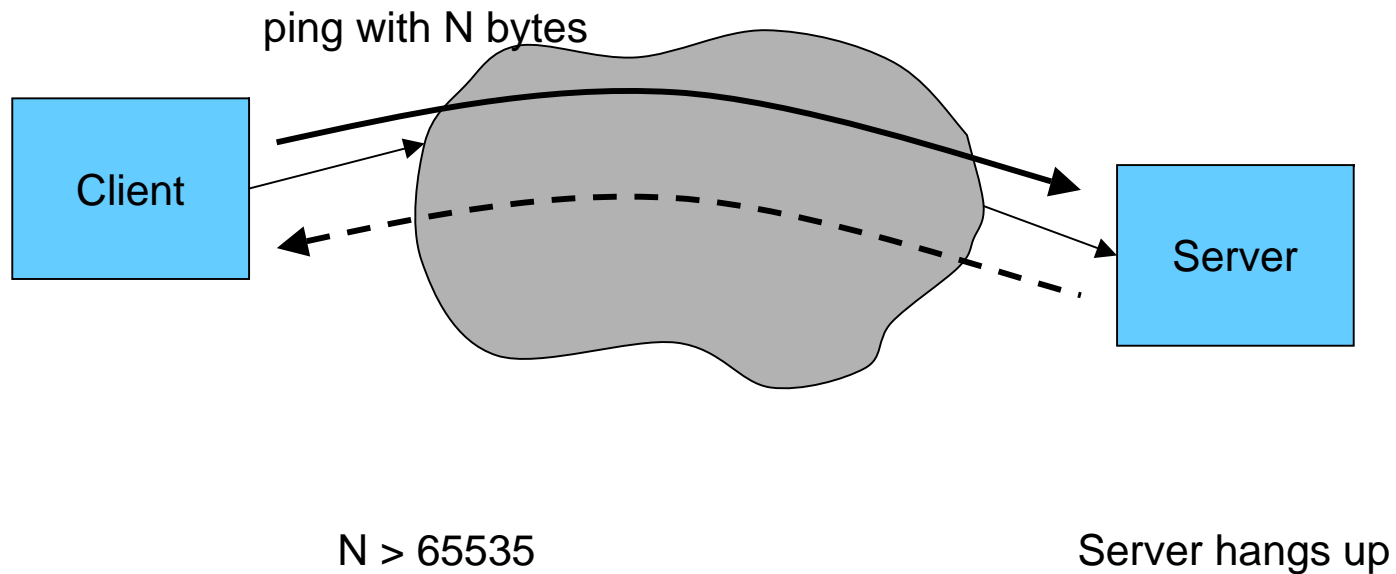
- If you are blocked from creating too many files in your own directory, get mail to do it for you in the mail directory.
- Send an endless string of messages to every user on the system. Has the side effect of making mail useless, maybe flooding the network with traffic as the mail gets bounced and forwarded and bounced some more.

Simple DoS



Simple DoS

The “Ping of Death”



Vulnerable Code

- Consider this (frequently occurring) C code segment:

```
void victim(parameters)
{
    int some_critical_data;
    char c[120];
    int more_critical_data;
    .
    .
    .
    while (*c++ = getc())
    {
        /* do something interesting */
    }
}
```

Vulnerable Code

- Consider this (frequently occurring) C code segment:

```
void victim(parameters)
{
    int some_critical_data;
    char c[120];                c is a fixed size array
    int more_critical_data;
    .
    .
    .
    while (*c++ = getc())      Keep reading characters
                                into c until an EOF occurs
    {
        /* do something interesting */
    }
}
```

Vulnerable Code

- Consider this (frequently occurring) C code segment:

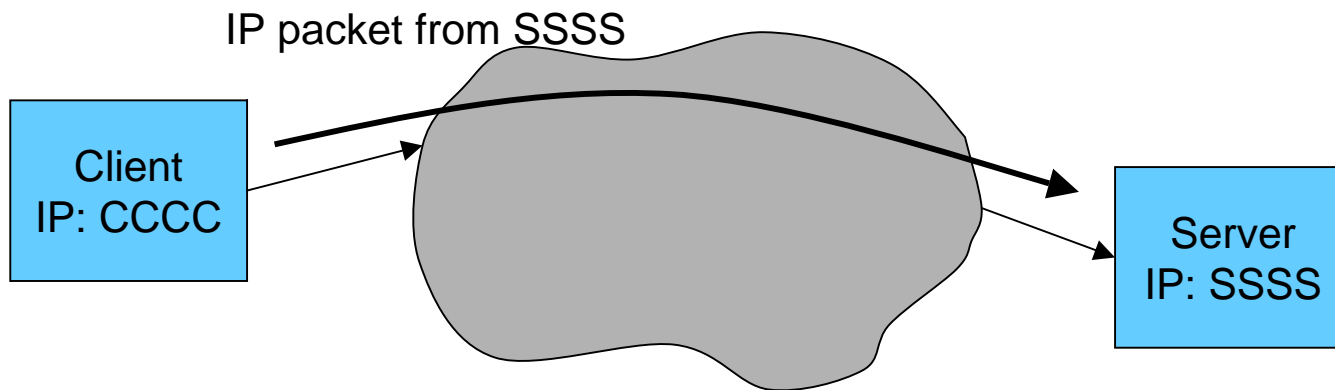
```
void victim(parameters)
{
  int some_critical_data;
  char c[120];
  int more_critical_data;
  .
  .
  .
  while (*c++ = getc())
  {
    /* do something interesting */
  }
}
```

c is a fixed size array

Keep reading characters into c until an EOF occurs

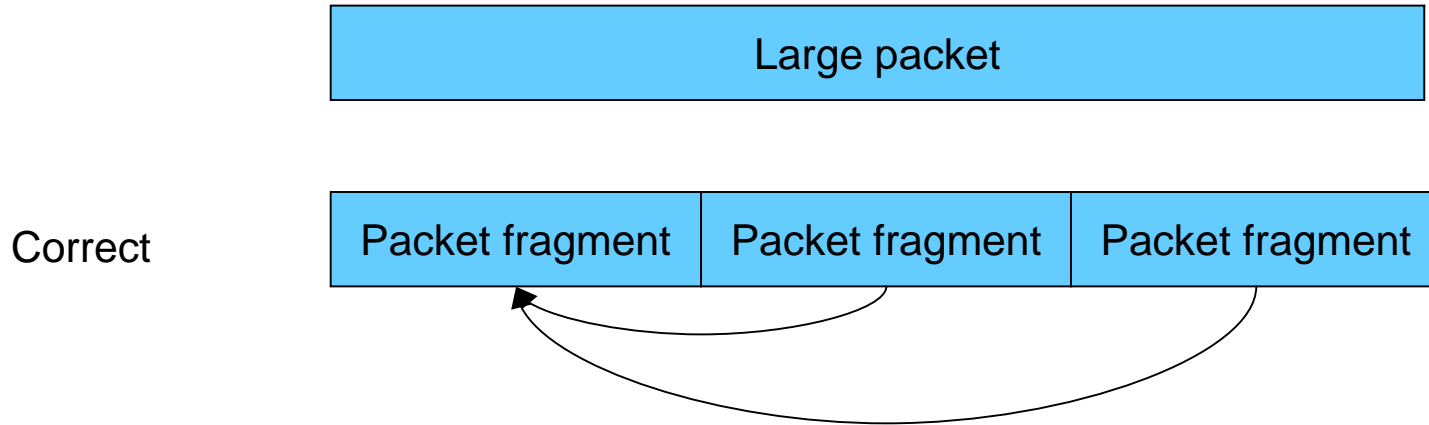
But, the 121st read overwrites critical data

Simple DoS – The Land Attack

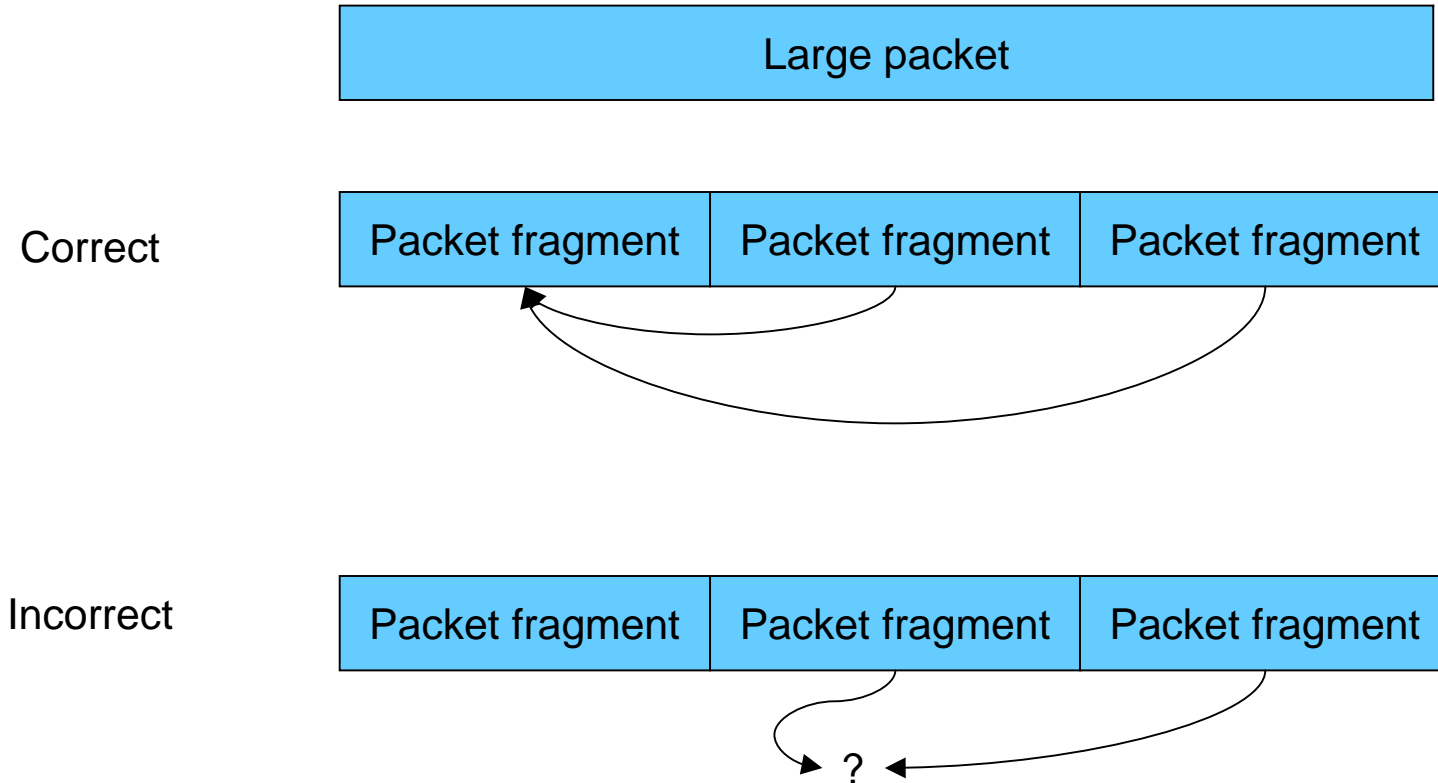


- Windows doesn't understand message and bogs down

Packet Fragmentation Errors

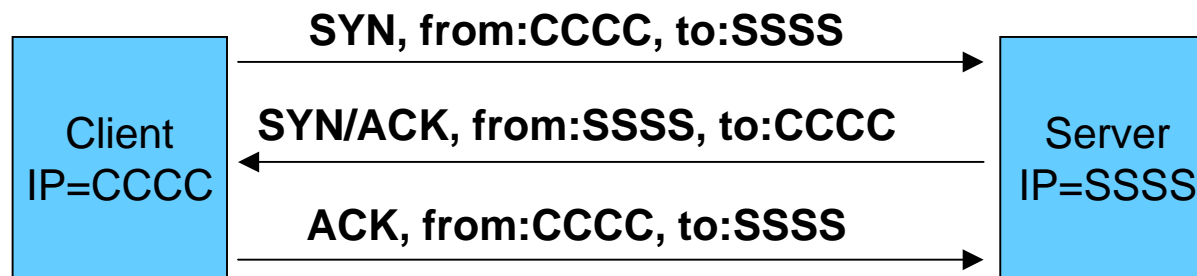


Packet Fragmentation Errors



How does receiving system handle improperly fragmented IP packets?
“teardrop” DoS attack

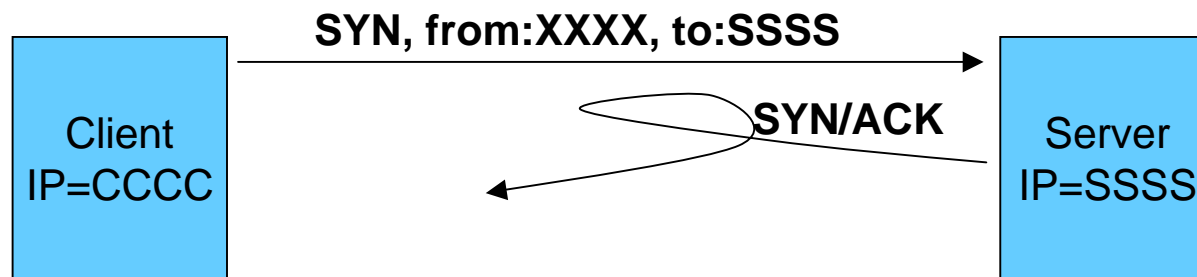
Anatomy of a TCP SYN DoS attack



1. Client process (e.g., NS) asks OS for a port
2. OS opens port in range 1024-65535
3. Client sends SYN to open path, including source/destination IP address
4. Client sends ACK to finish handshake

1. Server is listening on standard port in range of 1-1023, e.g., 80
2. Server receives SYN and responds with SYN/ACK, to ACK path and open reverse path. Server reserves resources for client (buffers, connection)

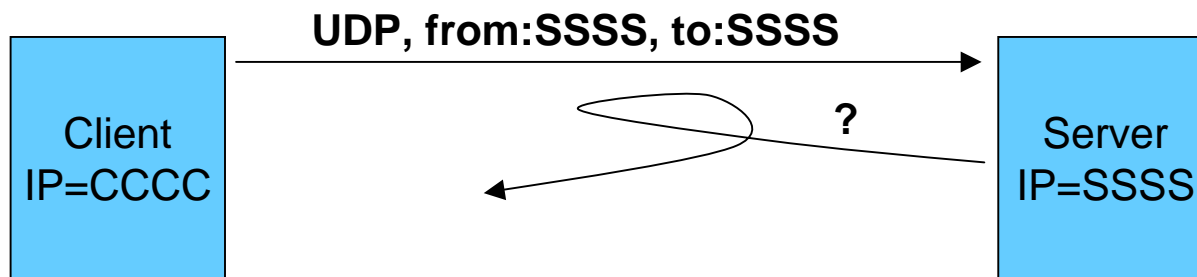
Old Style TCP DoS Attacks – The SYN Flood



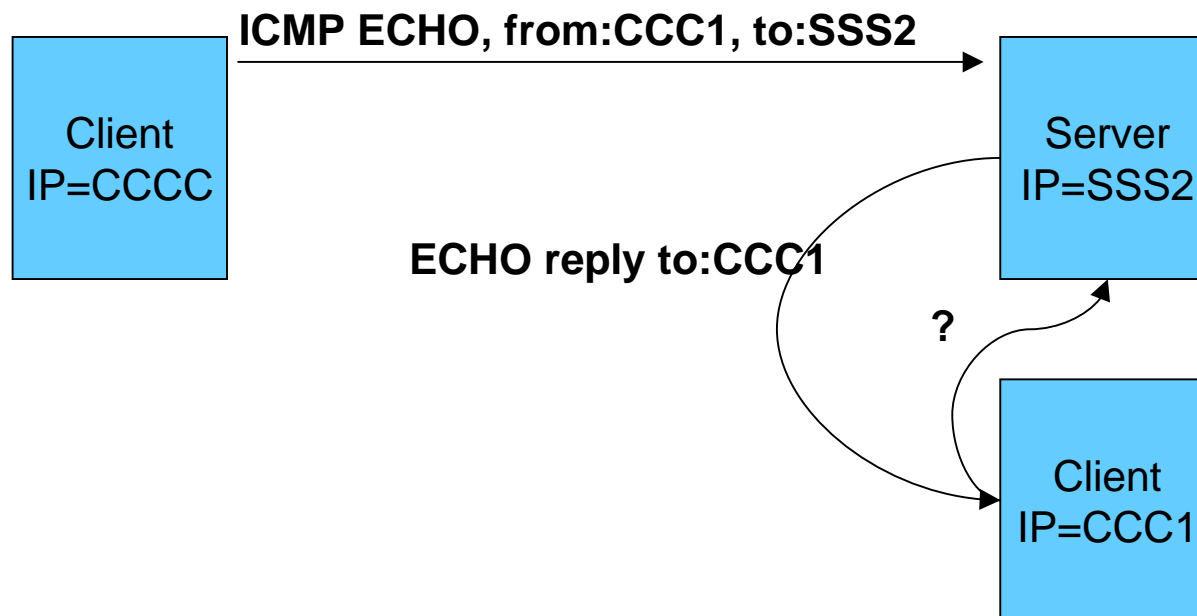
1. Client process (e.g., NS) asks OS for a port
2. OS opens port in range 1024-65535
3. Client sends SYN to open path, including bogus source IP address
4. Client keeps sending bogus SYNs to Server until Server cannot handle even valid connection requests

1. Server is listening on standard port in range of 1-1023, e.g., 80
2. Server receives SYN and responds with SYN/ACK, to ACK path and open reverse path. Server reserves resources for client (buffers, connection), but they are wasted.

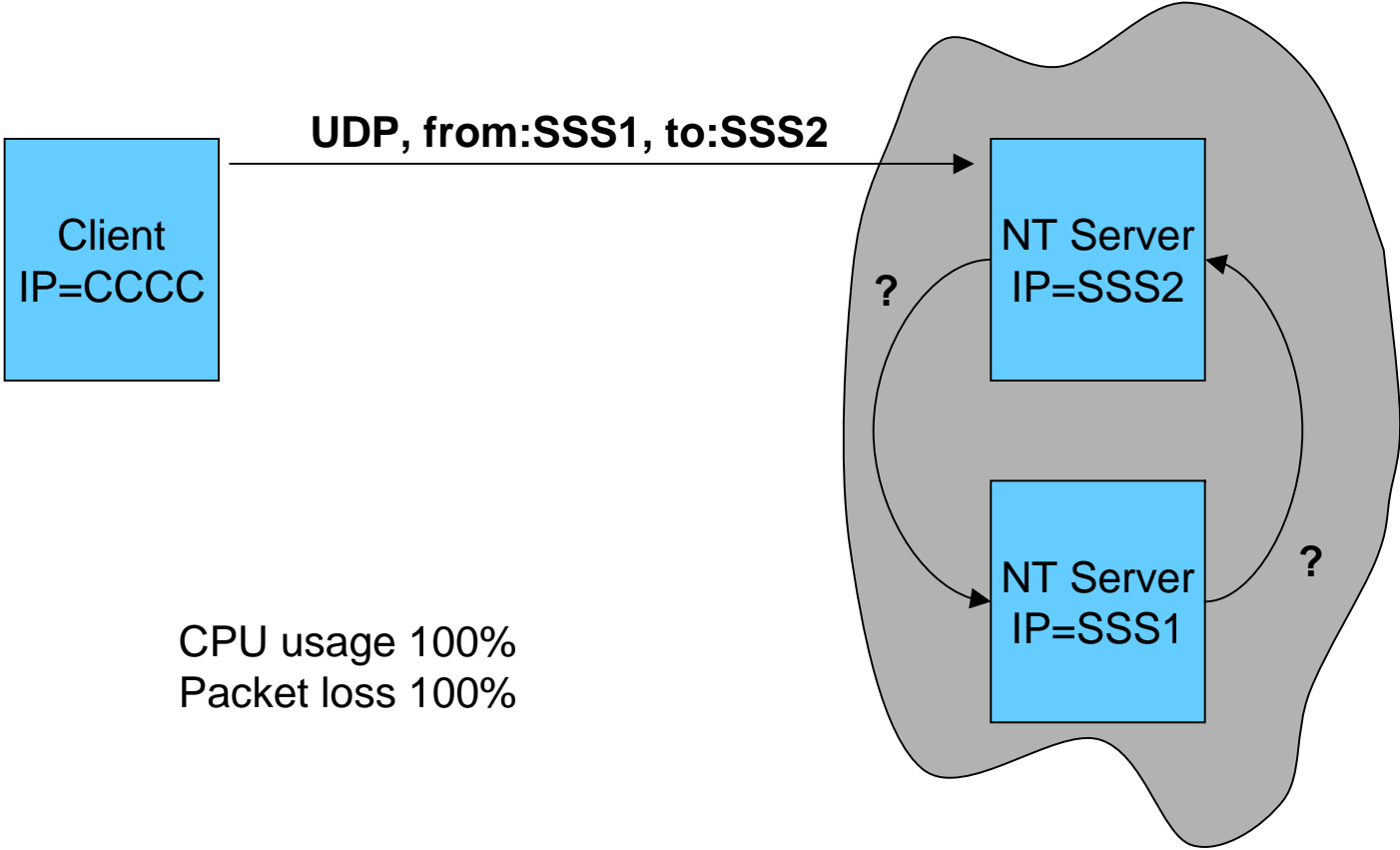
The UDP Flood



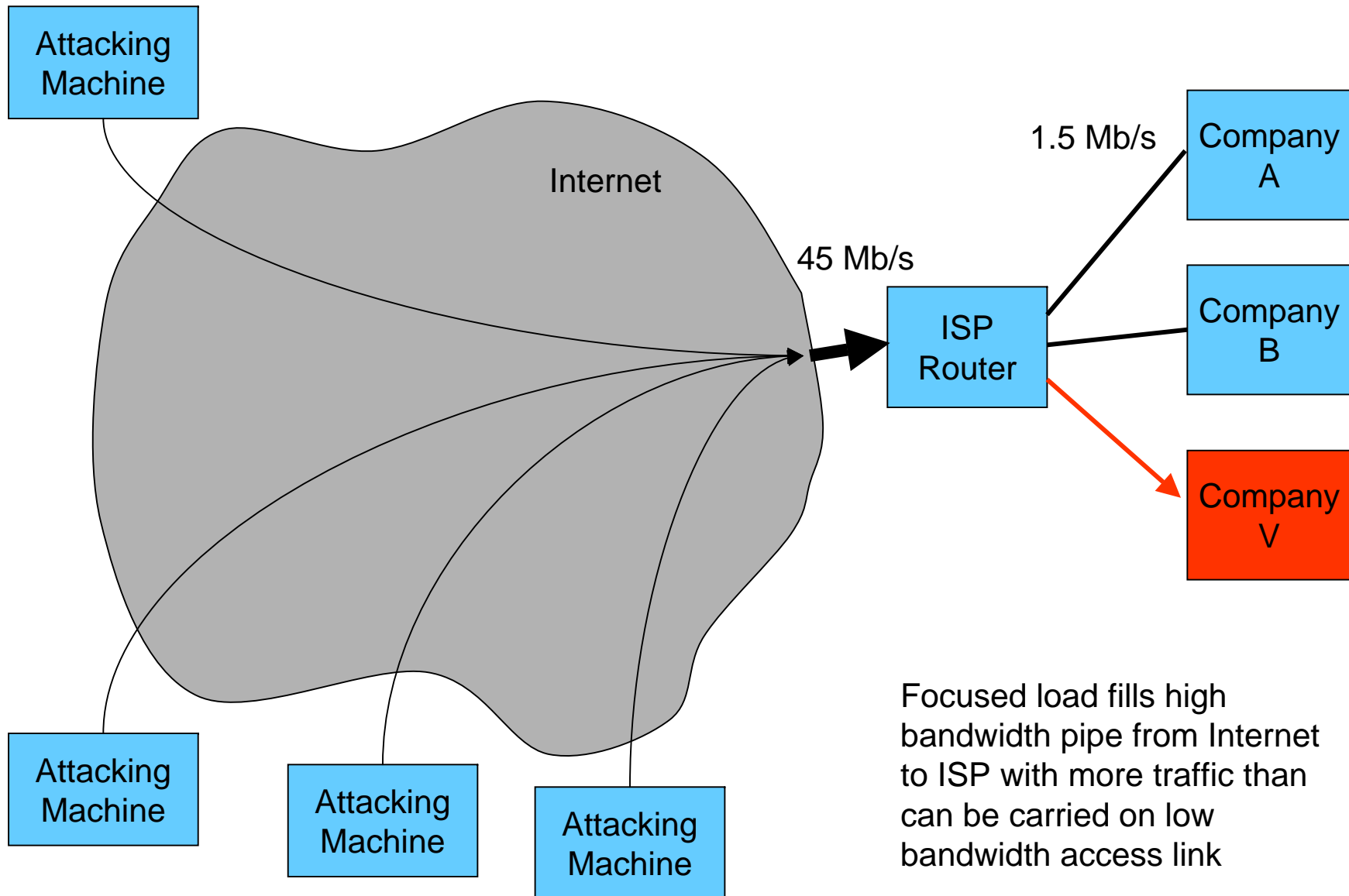
The ICMP ECHO Flood The Smurf Attack



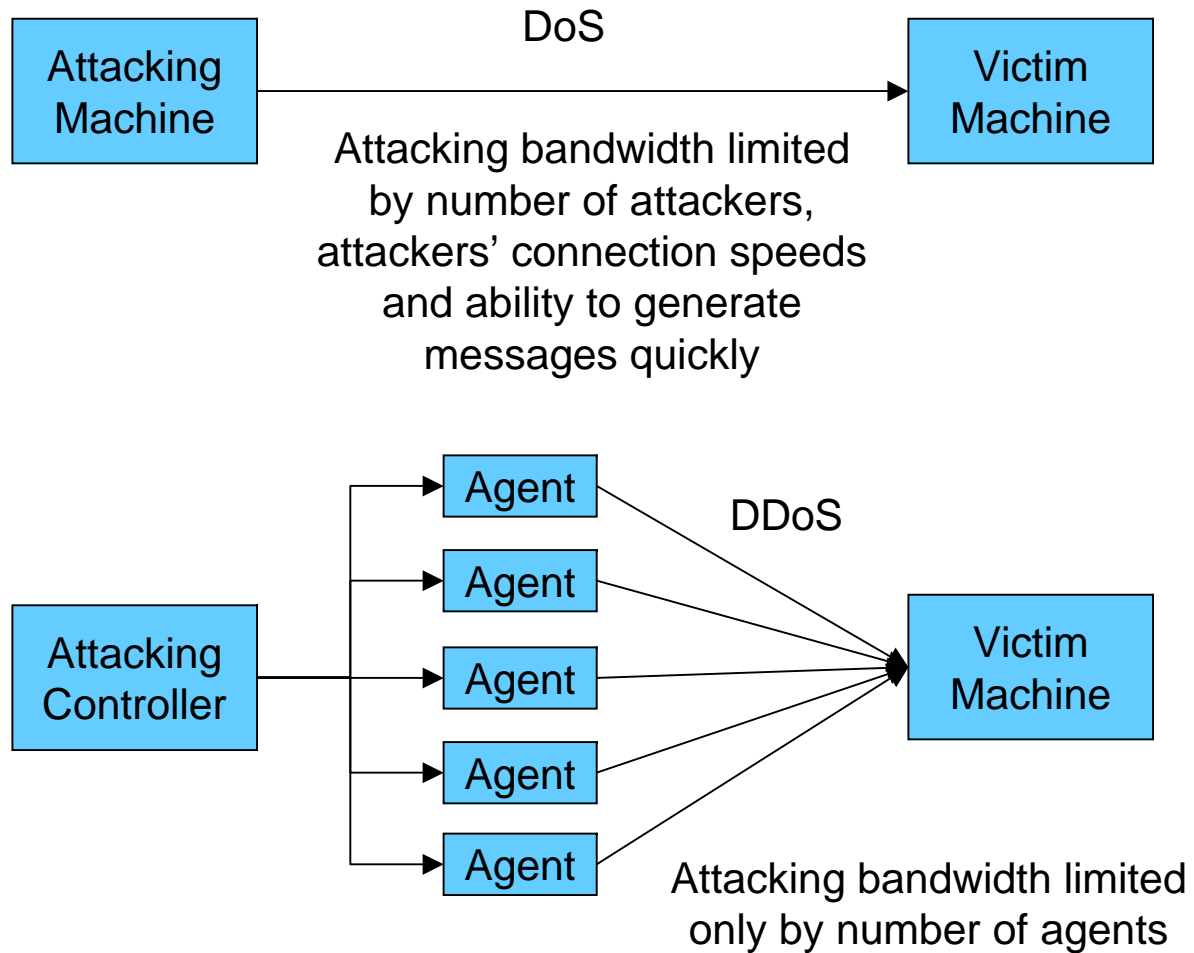
UDP Snork Attack



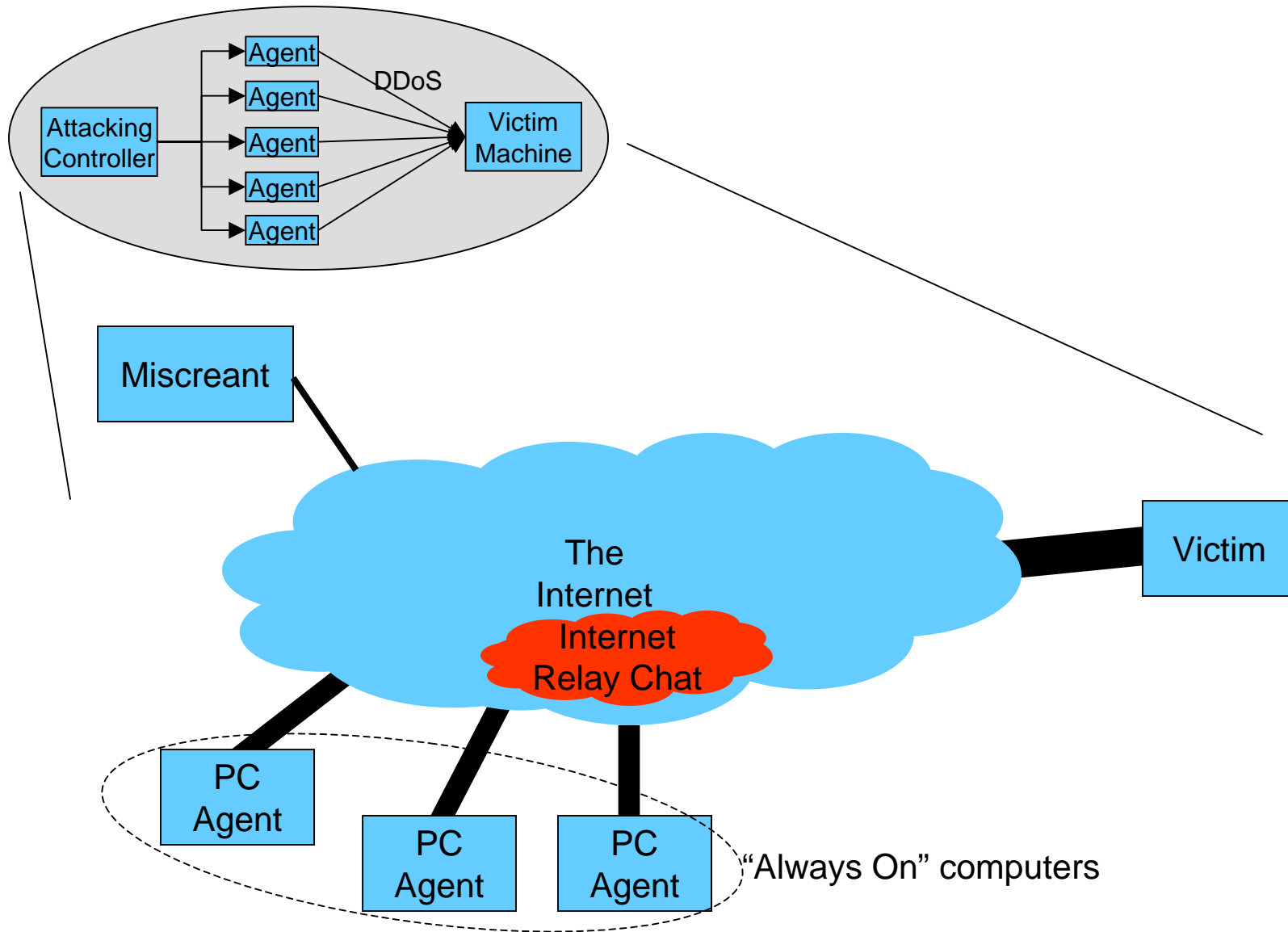
New Style DoS Attacks – DDoS



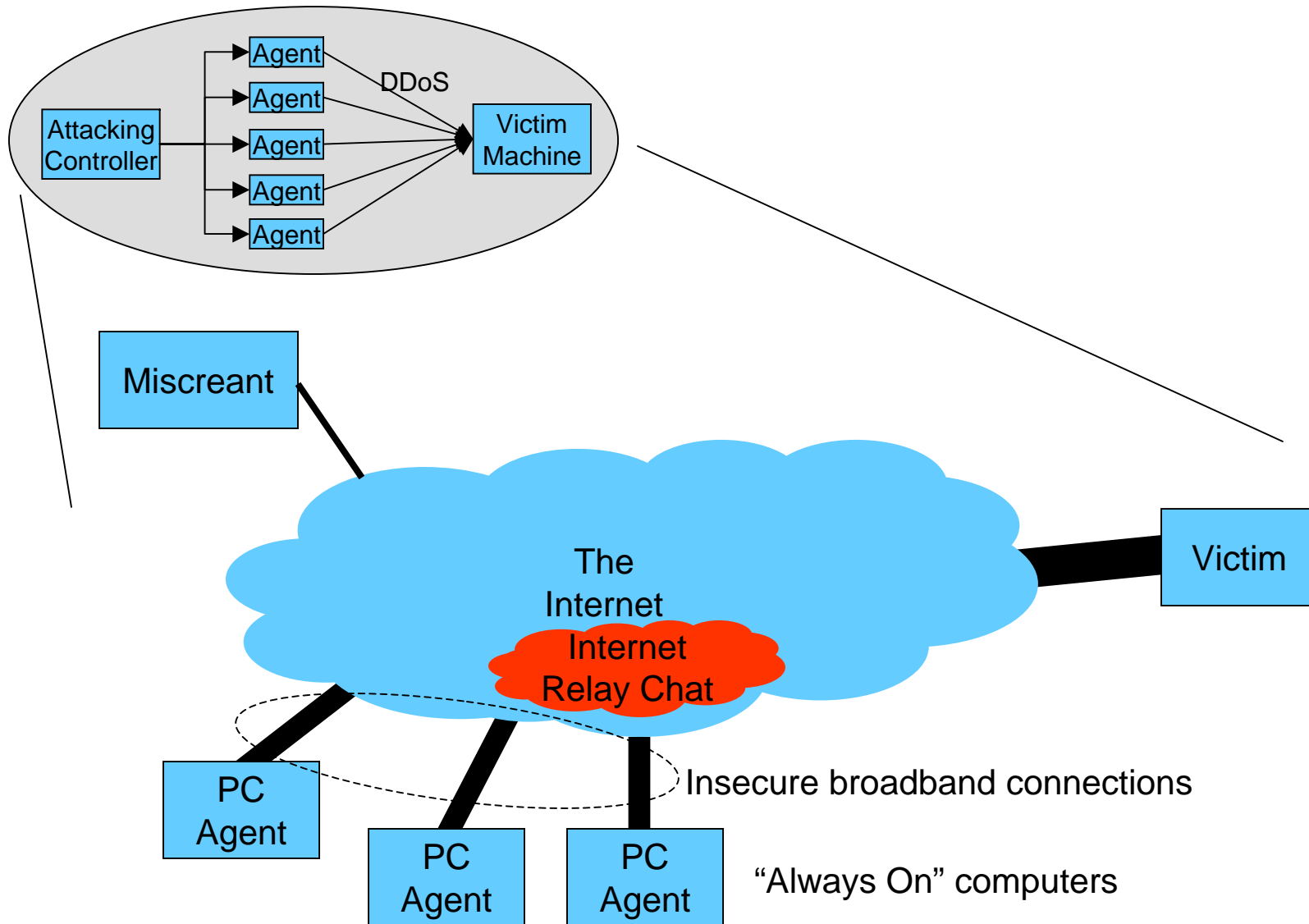
DoS vs. DDoS Attacks



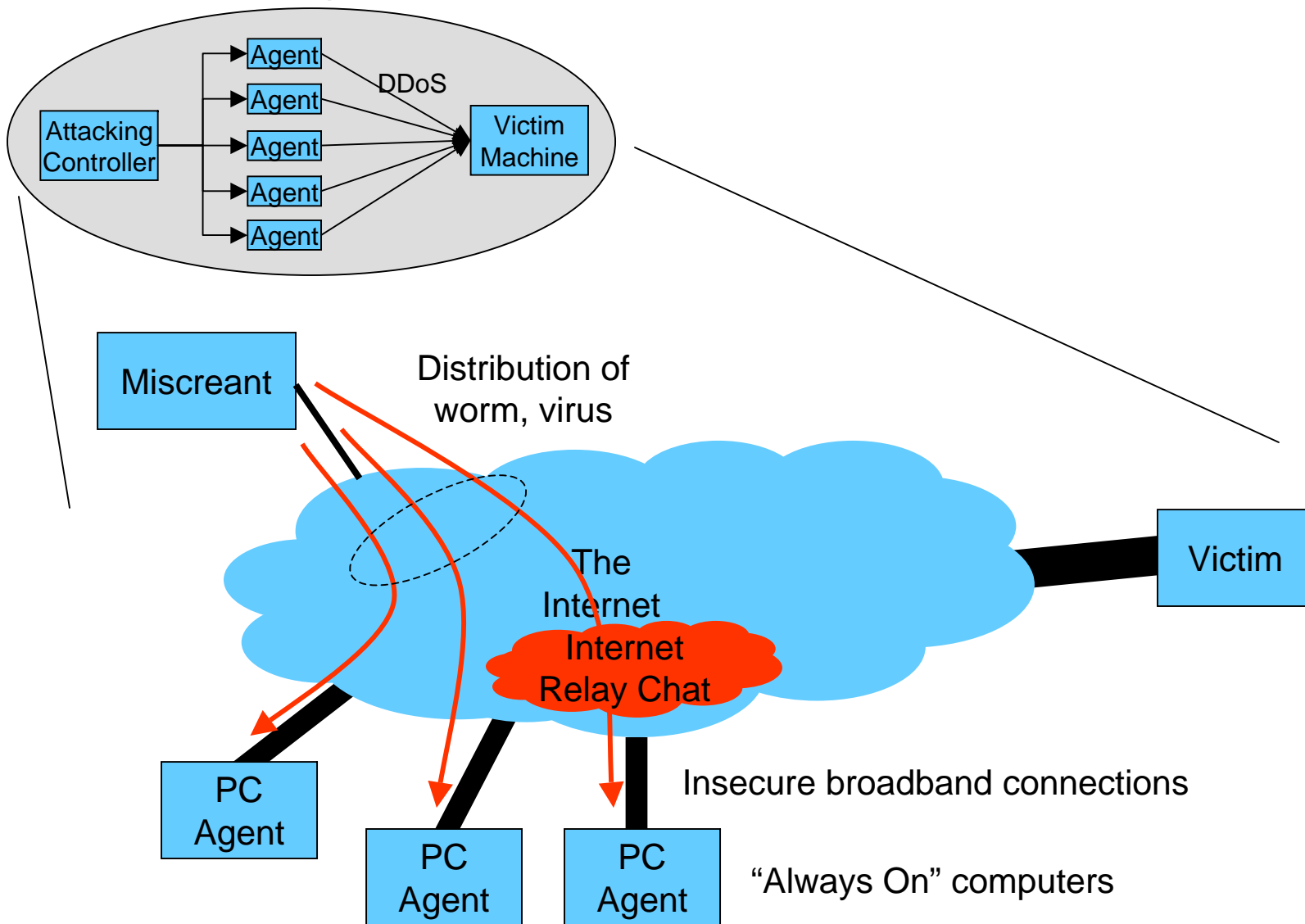
Getting "Zombies" to Help in DDoS Attacks



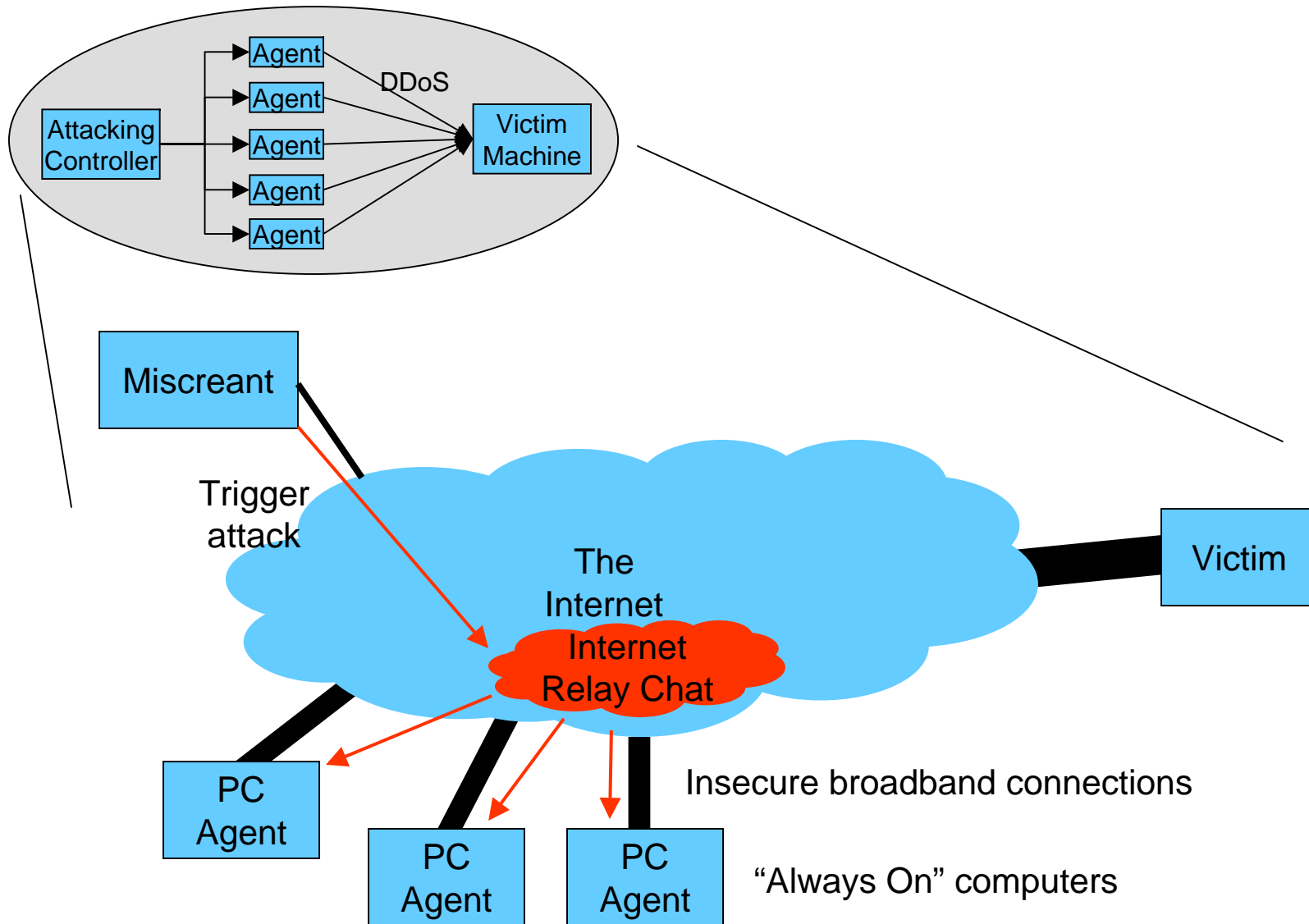
Getting "Zombies" to Help in DDoS Attacks



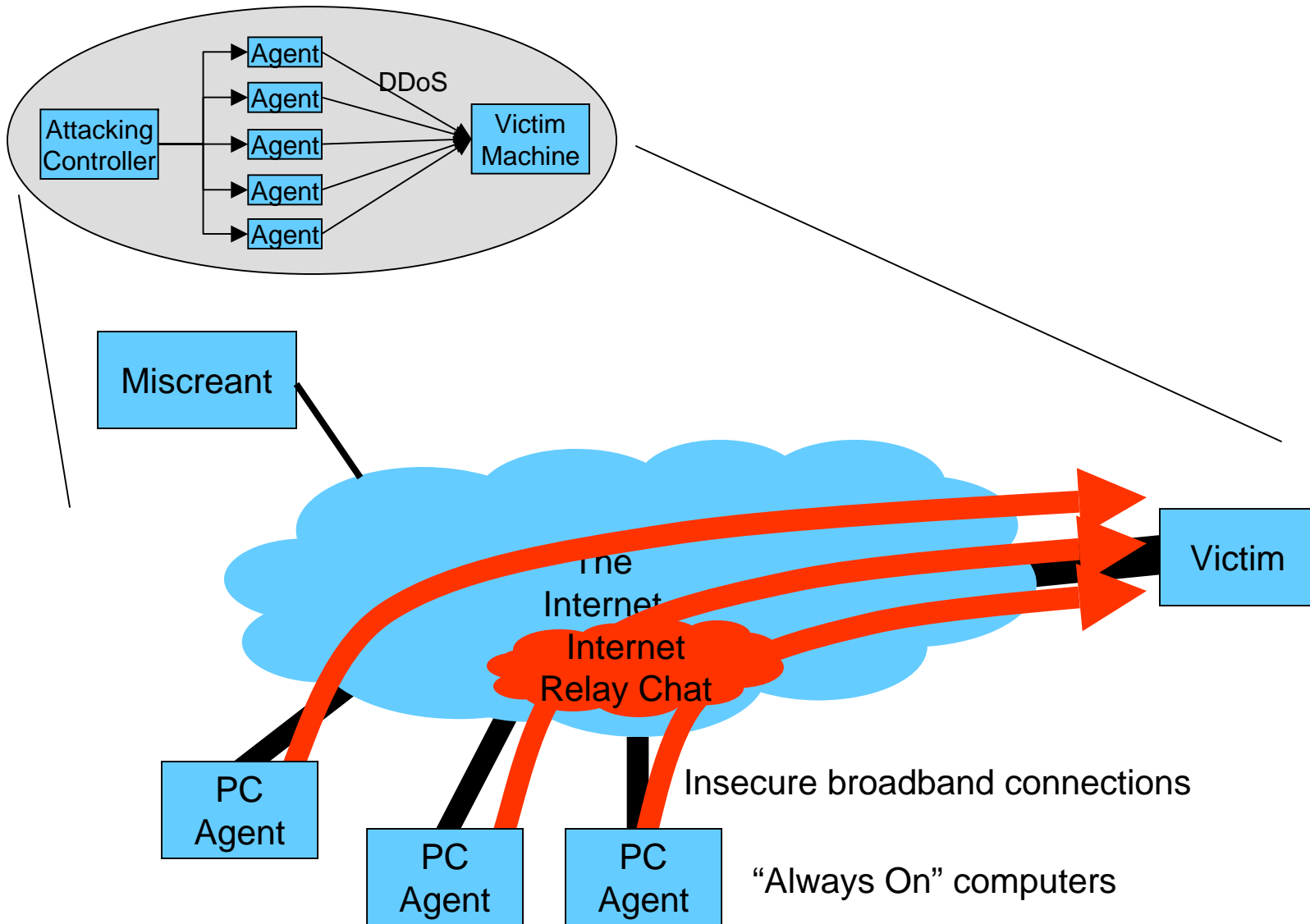
Getting "Zombies" to Help in DDoS Attacks



Getting “Zombies” to Help in DDoS Attacks

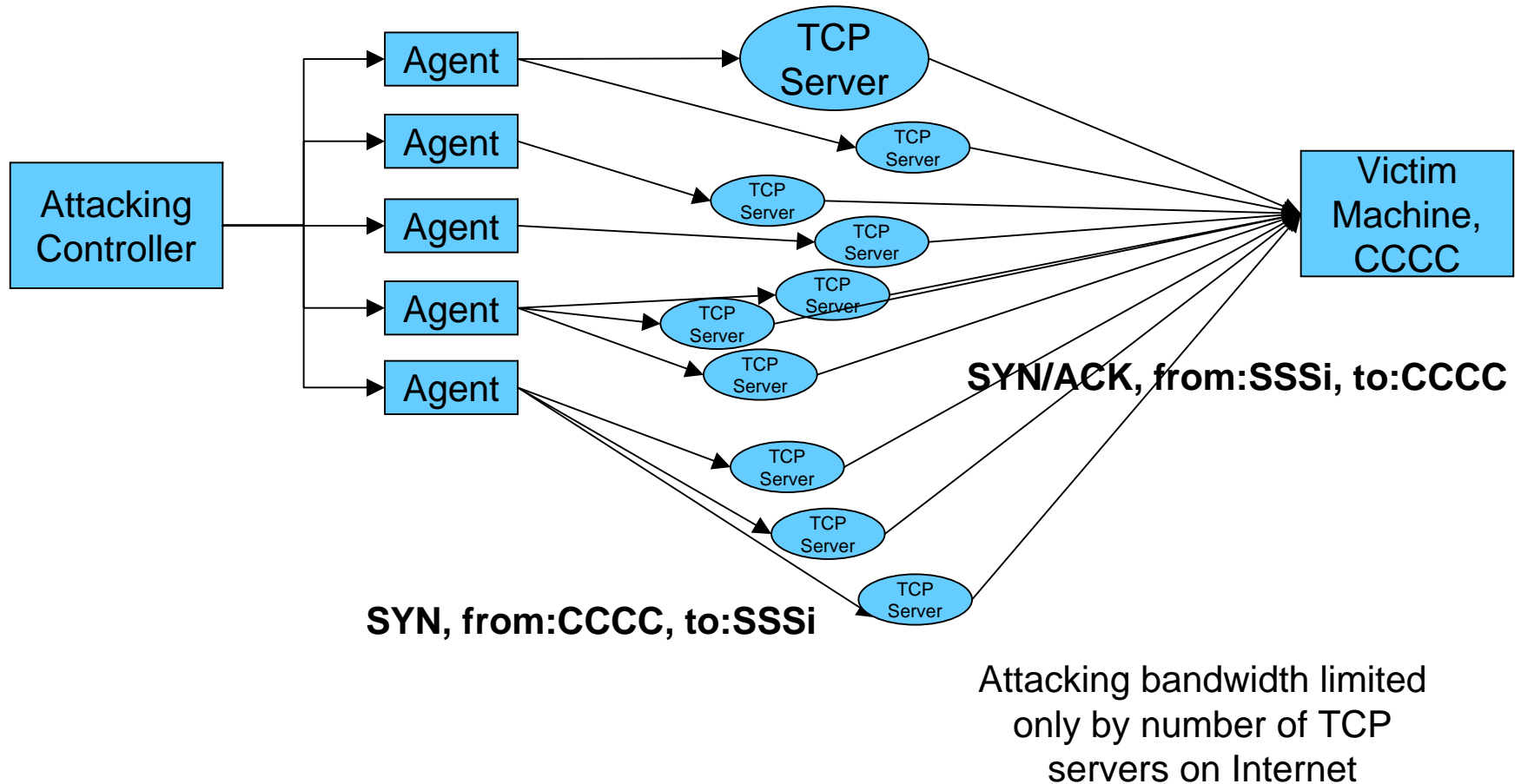


Getting "Zombies" to Help in DDoS Attacks



DRDoS Attacks

Using Reflections to Amplify Attack



Some Capabilities That Are Useful in Creating a Successful DoS Attack

- Exploitable system vulnerabilities
 - Obtaining root privileges on attack hosts
 - Ability to forge IP addresses (UNIX, linux, Win2k, WinXP, **not** Win98)
 - Ability to transmit packets with forged IP addresses through network
- Bounded resources
 - TCP connections
 - Connection bandwidth
 - memory
- Large number of easy to exploit attack vehicles
 - Home PCs connected via cable, DSL
- Anonymity
 - “Throwaway” dial-up PPP connections
- Inadequate security measures at *other* systems
 - Every unprotected system is a potential attack agent