Cooperative, Autonomous Anti-DDoS Network (A2D2V2)

Design and Implementation of a Cooperative, Autonomous Anti-DDoS Network using Intruder Detection and Isolation Protocol

Sarah Jelinek
Masters Project Defense
University of Colorado, Colorado Springs
sjjelinek@gmail.com
Committee:
Dr. C. Edward Chow
Dr. Jugal Kalita
Dr. Xiaobo Zhou

Outline

- Motivation and Goals for A2D2V2
- DoS and DDoS
  - What is it?
  - Mitigation strategies
- A2D2V2 Cooperative Detection and Mitigation Research
  - Cooperative Intrusion Response:
    - IDIP, CITRA, IDMEF, IDXL and CISL Protocols
  - Intrusion Detection – Dynamic Tracing
    - TCP link level headers, tcpdump
- A2D2 Overview

Outline

- A2D2V2
  - Features
  - Communication architecture
  - Implementation – IDIP components
- A2D2V2 Test Bed, Data Gathering and Results
- A2D2V2 Cooperative Defense Highlights
- A2D2V2 Conclusions
- Lessons Learned
- Future work recommendations

Motivation for A2D2V2

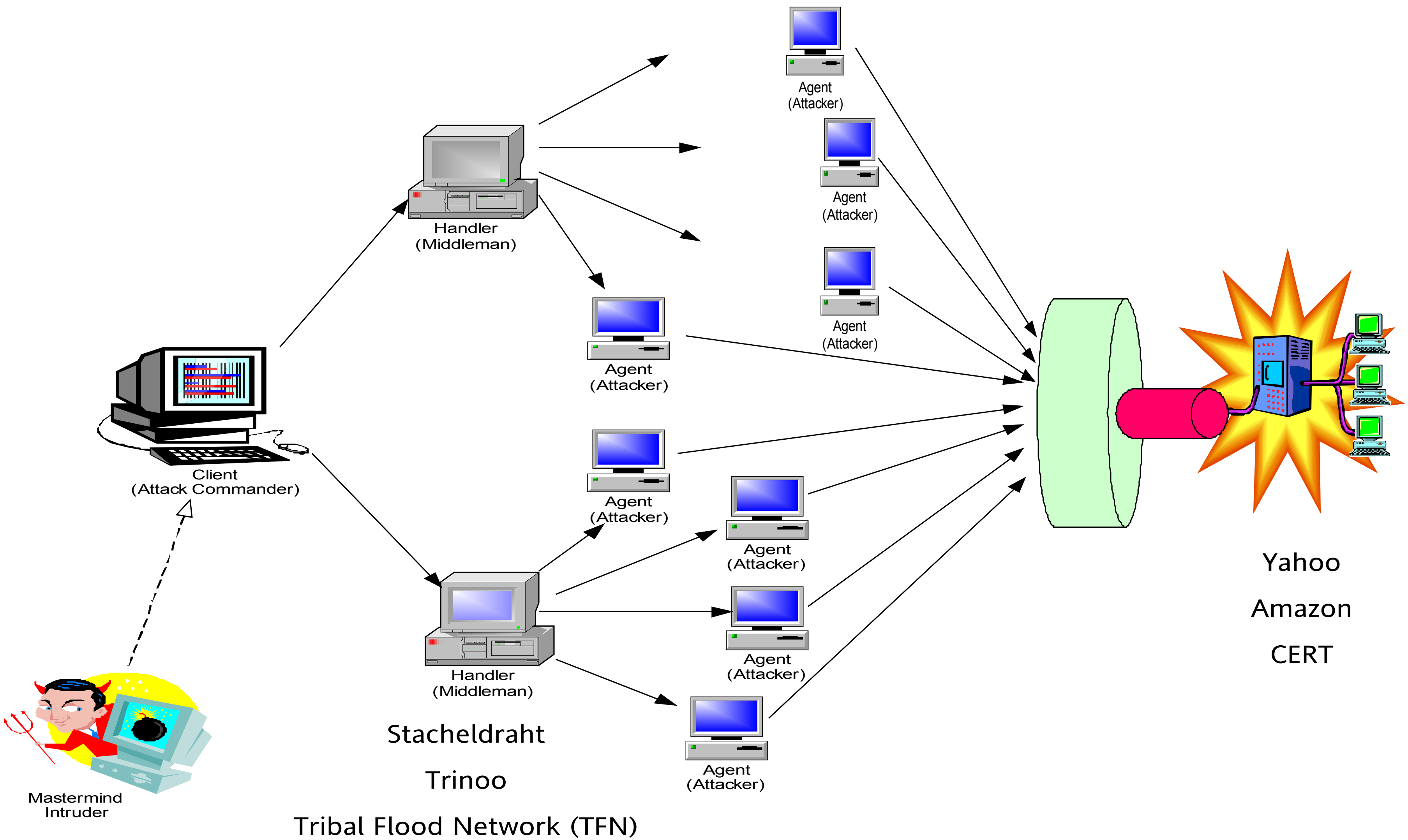
- DDoS and network security in general are still big areas of research
- Expand on initial A2D2 work
- No enterprise wide automated cooperative intrusion detection and response systems available

Goals for A2D2V2

- Expand on A2D2 ideas to provide cooperative defense against attacks
- To validate the enterprise effectiveness of the IDIP software implementation
- Show clients that are in non-IDIP enabled subnets reap benefits of enterprise network attack response cooperation
- Show that IDIP can provide a cooperative defense that efficiently notifies upstream routers of an attack

What is DoS/DDoS

- DoS – Denial of Service Attack
- DDoS Distributed Denial of Service Attack



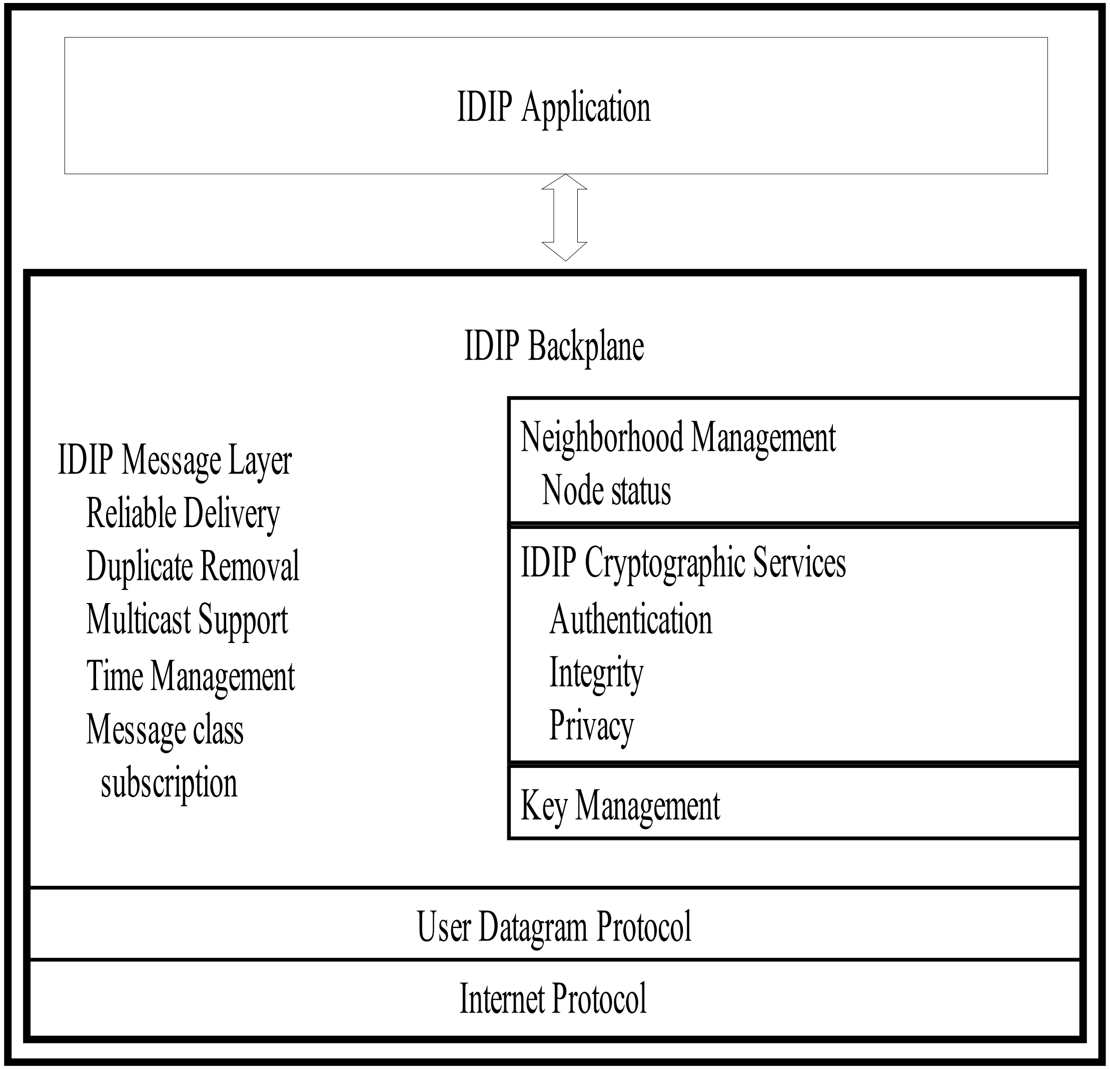
IDIP

- Intruder Detection and Isolation Protocol(IDIP)
  - Initially developed by DARPA, Boeing and NAI labs
  - Intended to be published, standard protocol. No longer open protocol.
  - Developed to support real-time tracking and containment of DDoS attacks that cross network boundaries. 2 stage response.
    - Initial response harsh and coarse grained,short lived
    - Subsequent response is more reasoned
  - Supports damage assessment and recovery in local environment
  - Provides network based response as well

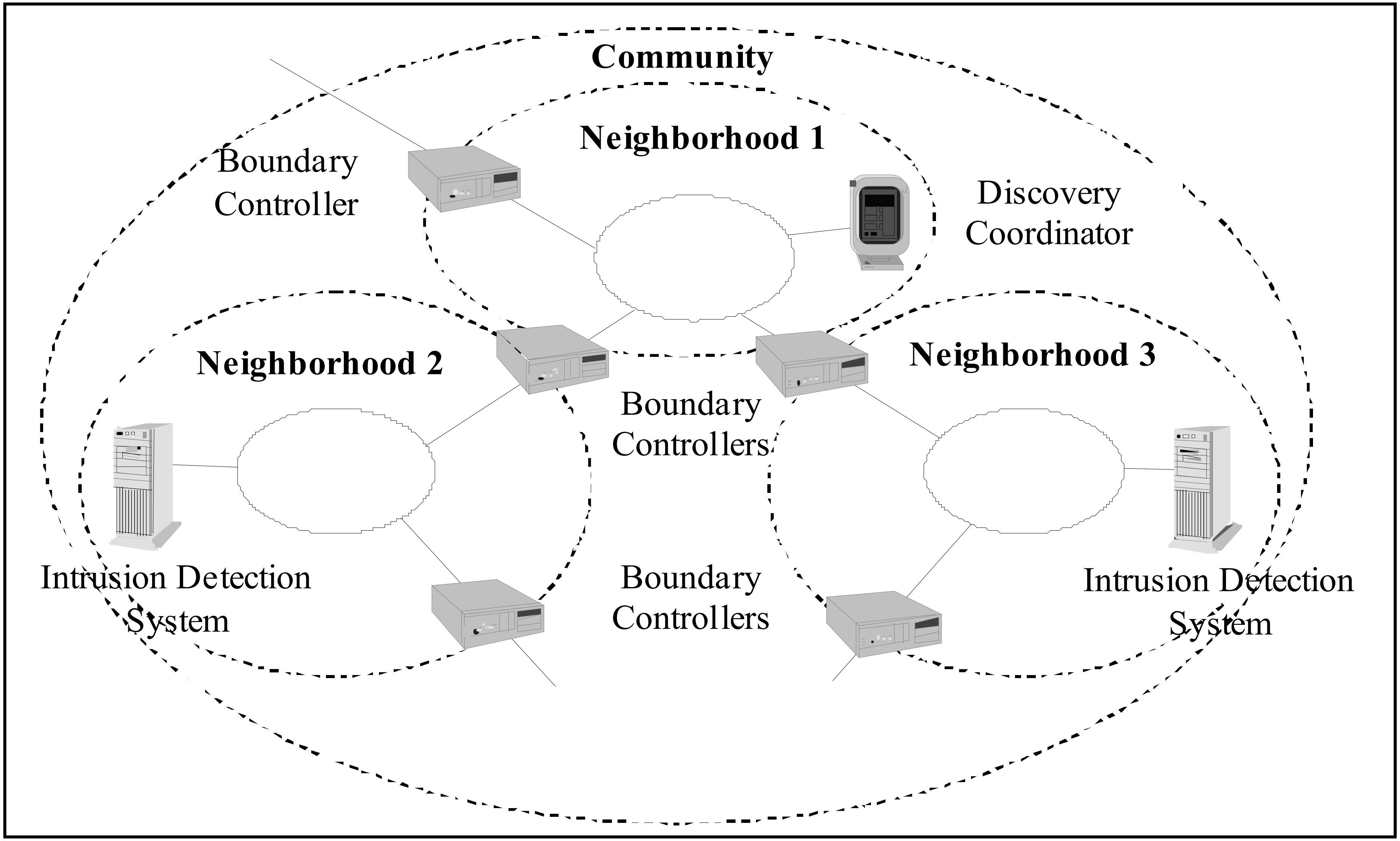
IDIP

- IDIP guiding principles
  - Response to intrusions in real-time
  - Support of environments that span multiple administrative domains
  - Minimal impact on systems performance
  - Autonomous & continued operation even under attack

IDIP Protocols and Layering



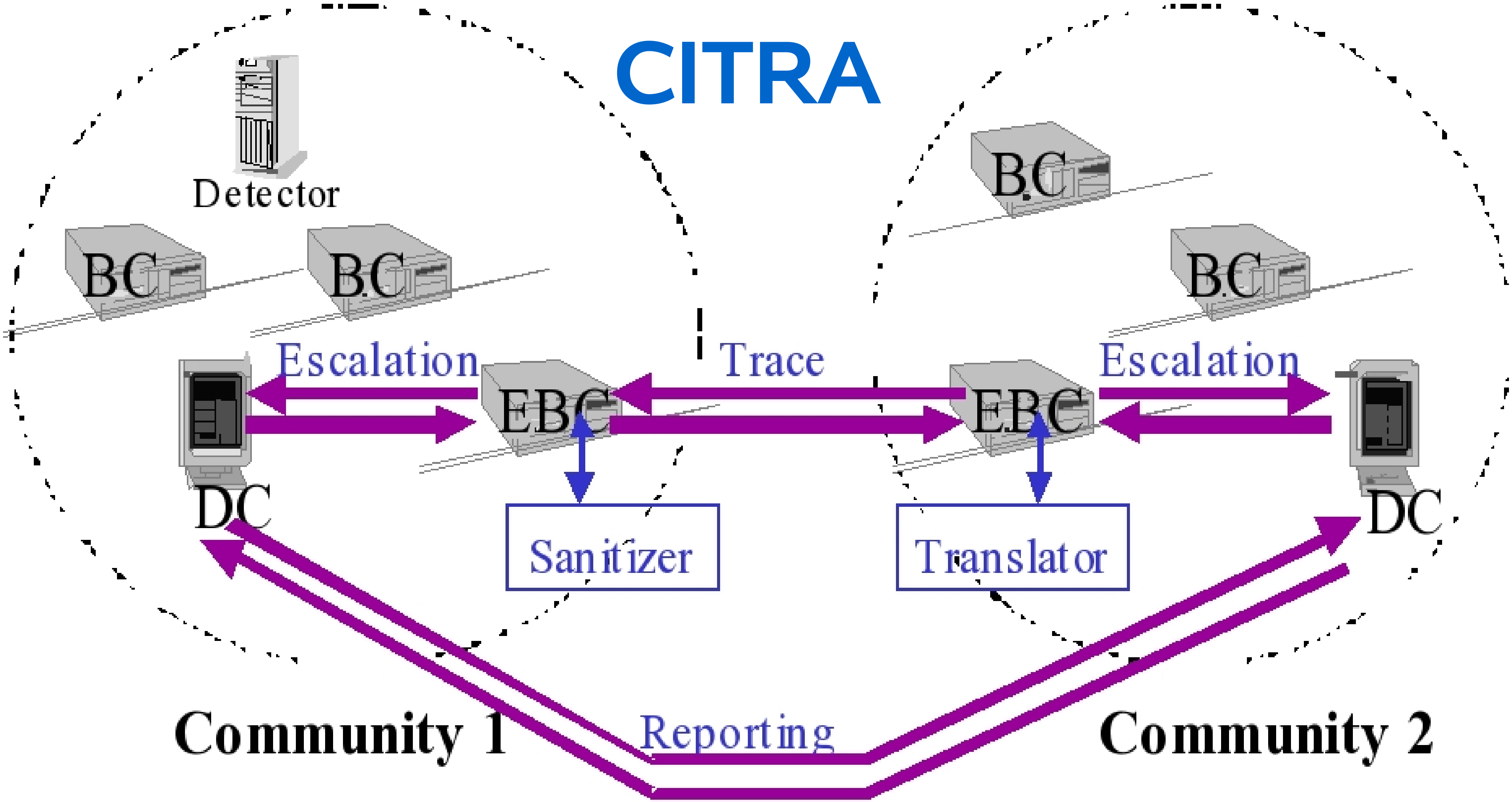
IDIP Enterprise Architecture



Cooperative Intrusion Detection Traceback Architecture, Common Intrusion Specification Language(CITRA and CISL)

- CITRA
  - Framework for integration of IDS, firewalls, routers, and other components in an IDIP system.
  - Allows for a global response via IDIP node cooperation
  - Designed to facilitate low-cost integration of independently developed components
  - IDIP defines the format of and information specification that CITRA enabled components exchange
- CISL
  - Language developed to support CITRA
  - Used to disseminate data among IDS and response systems

CITRA



DC - Discovery Coordinator (Management Console)

BC - Boundary Controller (Firewalls, Routers, etc.)

EBC - Edge Boundary Controller (e.g., Corporate Firewall)

Intrusion Detection Message Exchange Format(IDMEF)

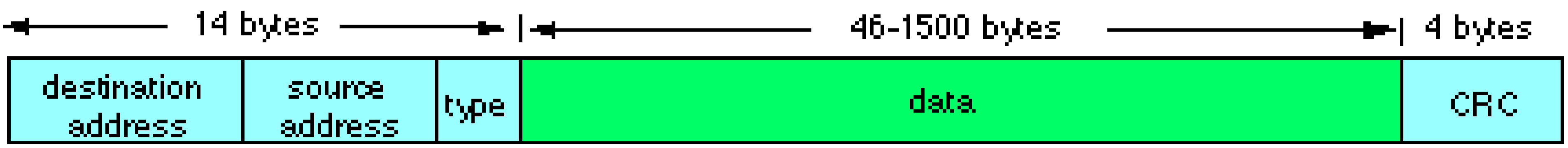
- Purpose to define formats and exchange procedures for sharing information
- Intended to standardize data format that automated IDS's can use to report alerts
- Enables interoperability among commercial and opensource IDS's.
- OO representation of alert data
- Data model allows for natural differences
- Goal is to provide a standardization of alerts in an unambiguous manner
- Implemented in XML

Intrusion Detection and Exchange Protocol(IDXP)

- Another protocol to exchange data between IDS entities
- Supports mutual authentication, integrity and confidentiality
- Provides for exchange of IDMEF messages, unstructured data between IDS systems
- Open, published standard

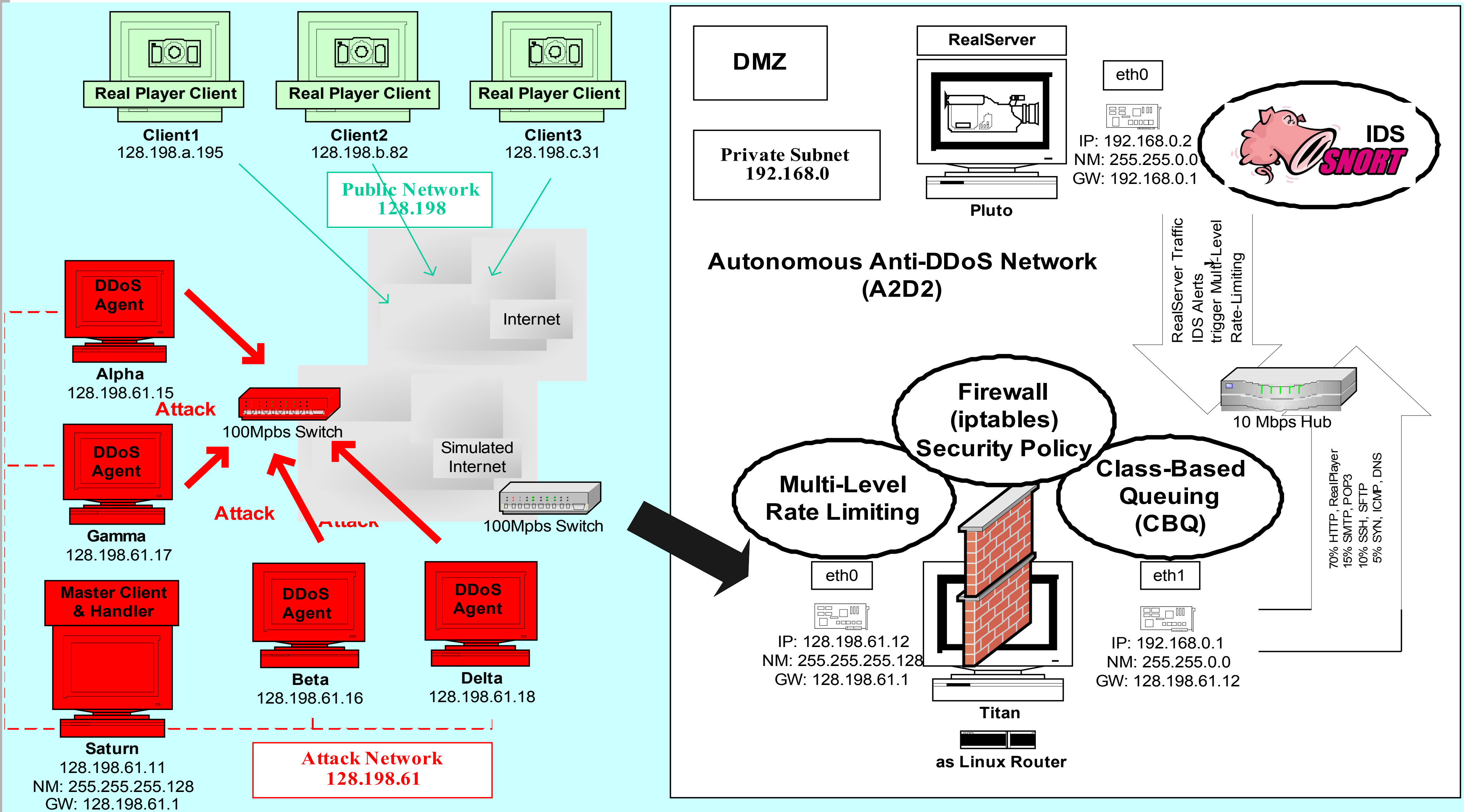
Dynamic Tracing

- IP Link Level Headers and ARP
  - Parsing the IP Packet link level header for MAC address



  - Use arp/rarp for resolving this to real IP address
  - ARP and RARP limitations
- tcpdump
  - Allows for fined grained control of monitoring interfaces
  - Is promiscuous

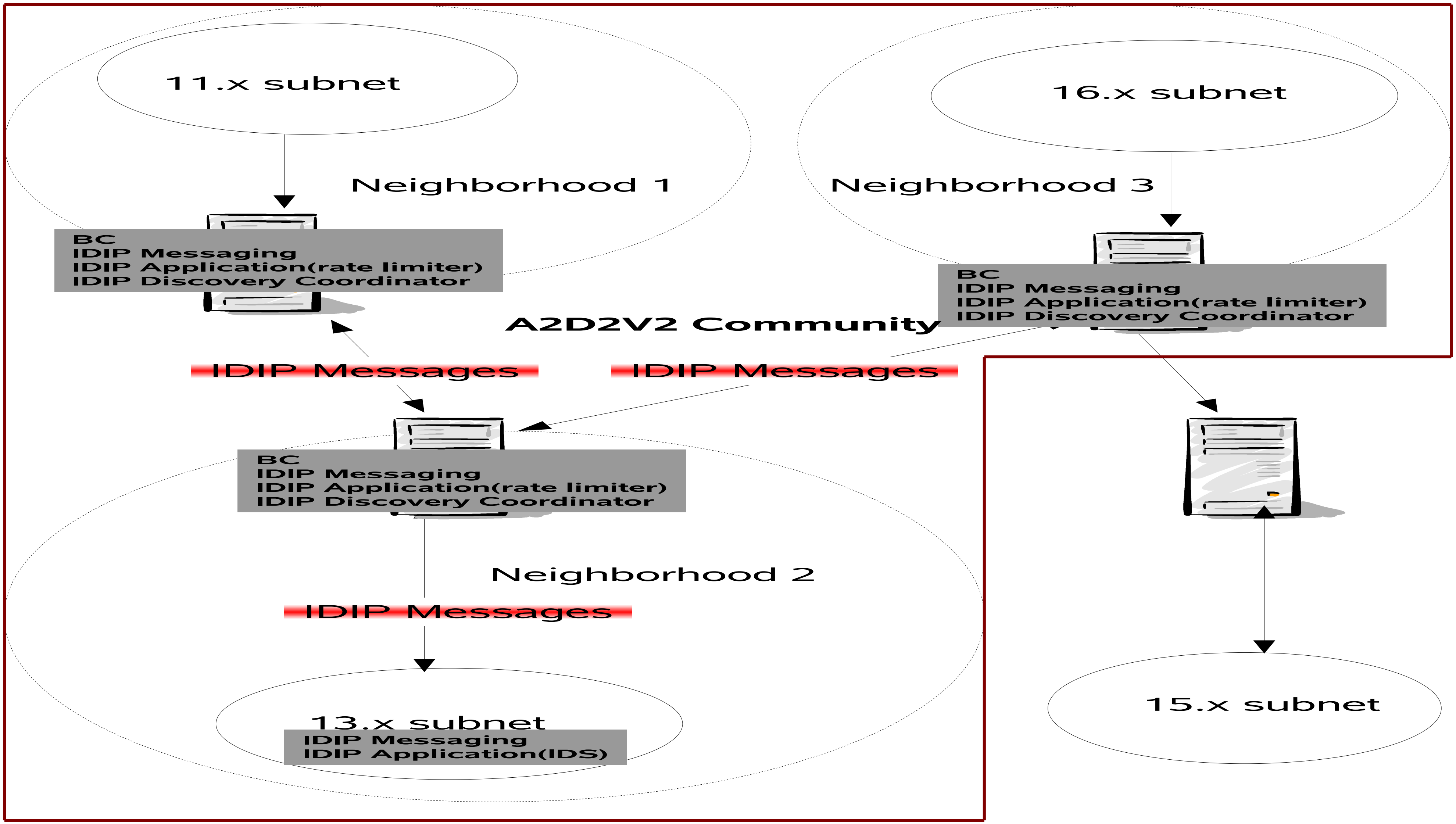
A2D2


Real Player Client
Client1
128.198.a.195
Real Player Client
Client2
128.198.b.82
Real Player Client
Client3
128.198.c.31
Public Network
128.198
Internet
DDoS Agent
Alpha
128.198.61.15
Attack
100Mpbs Switch
DDoS Agent
Gamma
128.198.61.17
Attack
Simulated Internet
100Mpbs Switch
Master Client & Handler
Saturn
128.198.61.11
NM: 255.255.255.128
GW: 128.198.61.1
DDoS Agent
Beta
128.198.61.16
DDoS Agent
Delta
128.198.61.18
Attack Network
128.198.61
DMZ
Private Subnet
192.168.0
RealServer
Pluto
eth0
IP: 192.168.0.2
NM: 255.255.0.0
GW: 192.168.0.1
IDS
SNORT
Autonomous Anti-DDoS Network
(A2D2)
RealServer Traffic
IDS Alerts
trigger Multi-Level
Rate-Limiting
10 Mbps Hub
Firewall
(iptables)
Security Policy
Multi-Level
Rate Limiting
Class-Based
Queuing
(CBQ)
eth0
IP: 128.198.61.12
NM: 255.255.255.128
GW: 128.198.61.1
Titan
as Linux Router
eth1
IP: 192.168.0.1
NM: 255.255.0.0
GW: 128.198.61.12
70% HTTP, RealPlayer
15% SMTP, POP3
10% SSH, SFTP
5% SYN, ICMP, DNS

A2D2V2 Features

- 7 key feature additions from A2D2
  - IDIP Additions to Snort IDS
    - report_idip and preprocessor changes
  - IDIP Enabled firewall/routers
    - idip_firewall_receiver
  - Earlier detection and pushback of attack via traffic monitoring
    - tcpdump.sh, dumper.sh awk scripts
  - Notification of upstream routers of attack
    - Static router configuration table
  - Notification to upstream routers of attack mitigation strategies taken by surrounding neighborhoods and subsequent response
    - Response policy is accept

A2D2V2 IDIP Communication and Neighborhoods Design



A2D2V2 IDIP Modifications

- IDIP Messaging Protocol
  - IDIP Neighborhood management via the DC
  - Message creation and formatting
  - Protocol initialization
  - Message forwarding
  - Socket communication pieces
- IDIP Application Protocol
  - Snort modifications for IDIP support
  - IDIP enabled firewall/router application

A2D2V2 IDIP Communication Flow

Snort IDS ->generates flood report when attack is detected
report_idip -> intercepts flood report message
report_idip->creates three classes of IDIP messages:
IDIP DO
IDIP UNDO
IDIP TRACE
report_idip->forwards IDIP message to next immediate upstream firewall/router
idip_firewall_receiver->receives IDIP message and processes according to request

A2D2V2 IDIP Communication Flow

idip_firewall_receiver -> either:

performs trace using tcpdump

performs do(applies rate limiting to itself)

performs undo(undoes rate limiting as per request

notifies upstream routers of mitigation action taken

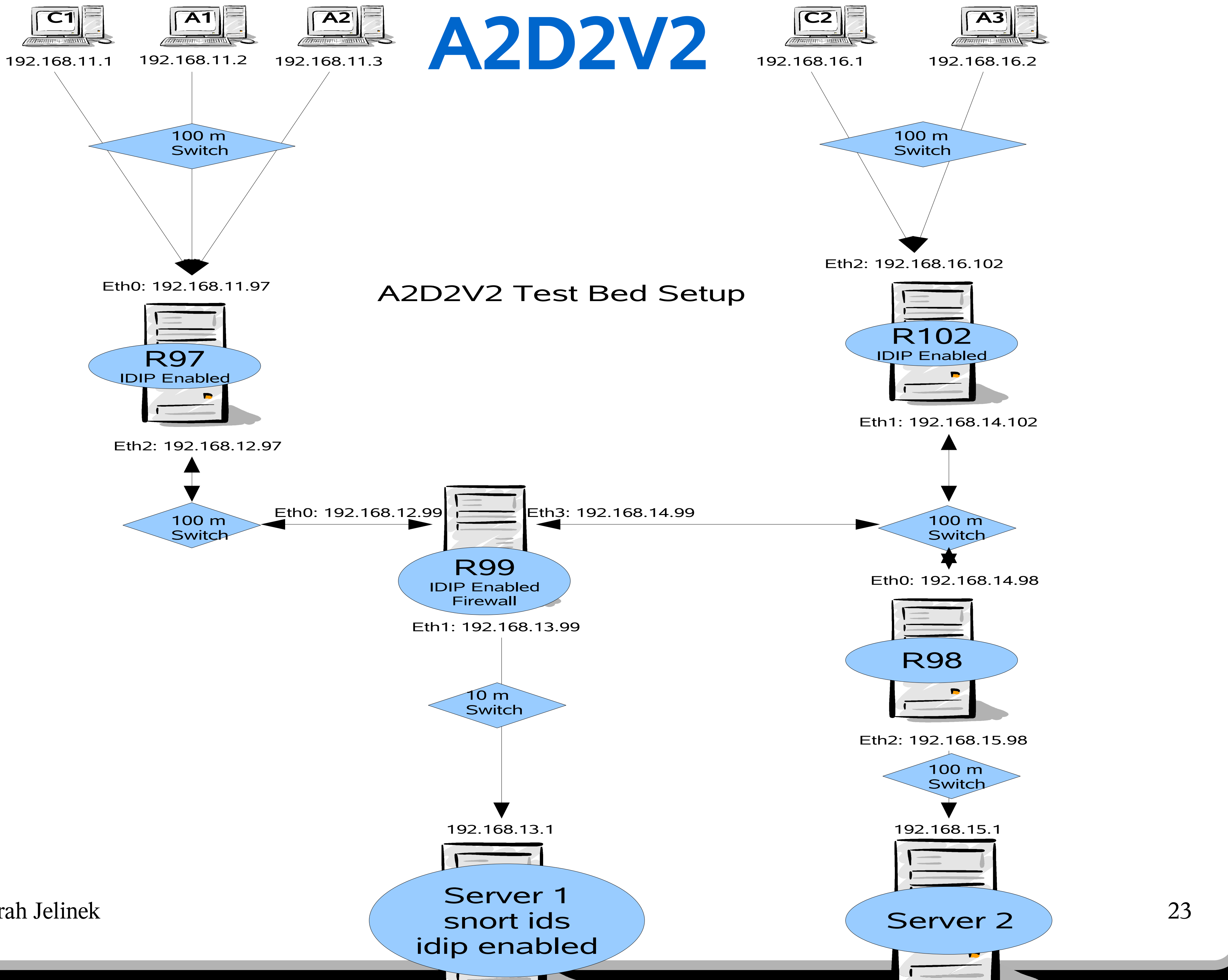
Recommends same action to be taken by upstream routers

idip_firewall_receiver on upstream router applies recommended action of rate limiting

A2D2V2 Implementation

- Key software modules:
  - firewall/routers:
    - idip_firewall_receiver – IDIP Application and Message Subsystem
    - tcpdump.sh, dumper.sh – IDIP Application
    - trace_kill – IDIP Application
    - topo.txt – DC Static configuration tables
    - A2D2 class based queueing and rate limiter modules
  - Server:
    - Snort with spp_flood preprocessor
    - report_idip – IDIP Application and Message subsystem
    - tcp_snd
  - Client:
    - tcp_rcv
    - A2D2 attack tool and packet counting modules

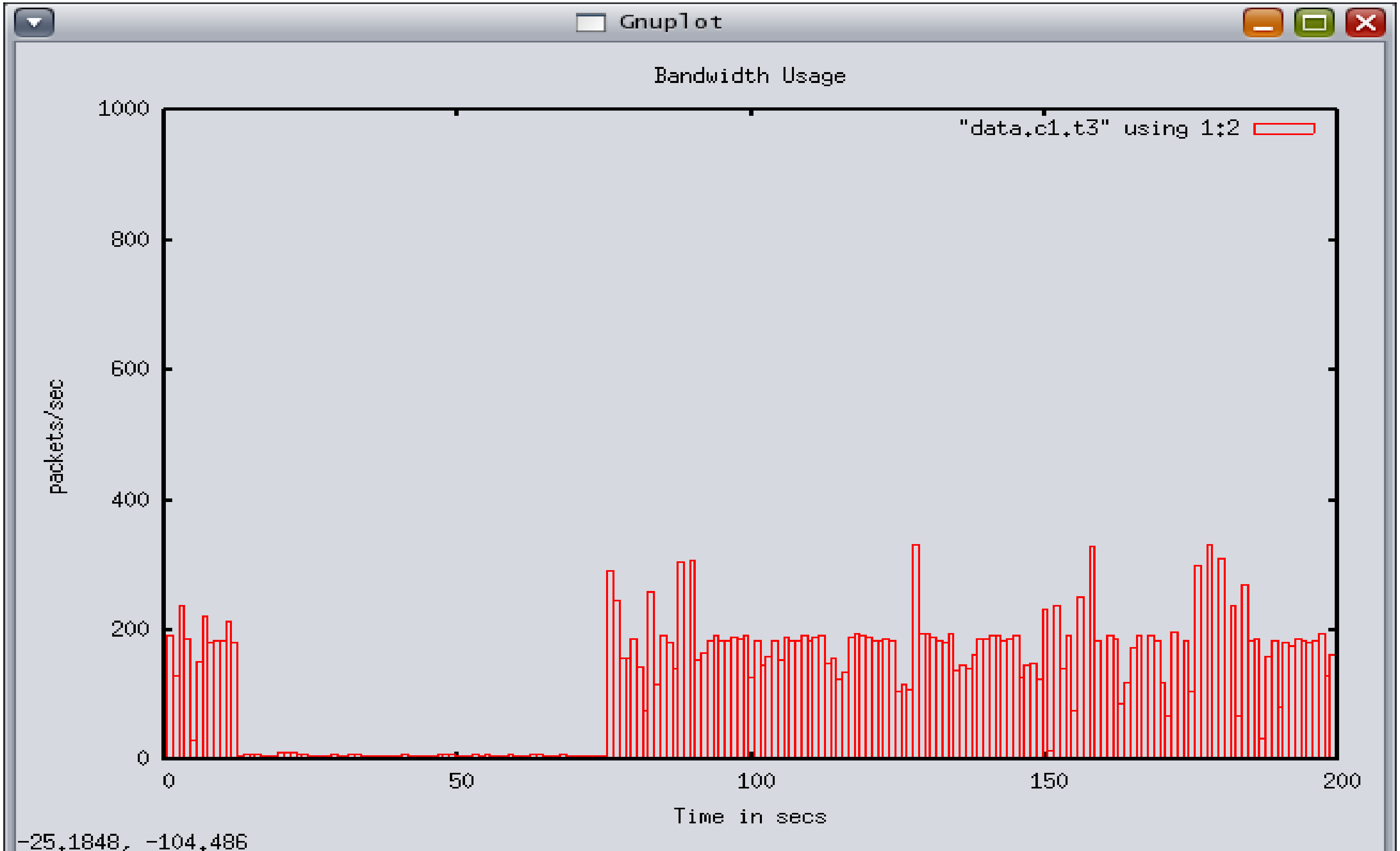
A2D2V2


C1
A1
A2
192.168.11.1
192.168.11.2
192.168.11.3
C2
A3
192.168.16.1
192.168.16.2
100 m Switch
100 m Switch
Eth2: 192.168.16.102
Eth0: 192.168.11.97
A2D2V2 Test Bed Setup
R102
IDIP Enabled
R97
IDIP Enabled
Eth1: 192.168.14.102
Eth2: 192.168.12.97
Eth0: 192.168.12.99
Eth3: 192.168.14.99
100 m Switch
100 m Switch
R99
IDIP Enabled
Firewall
Eth0: 192.168.14.98
Eth1: 192.168.13.99
R98
10 m Switch
Eth2: 192.168.15.98
100 m Switch
192.168.13.1
192.168.15.1
Server 1
snort ids
idip enabled
Server 2

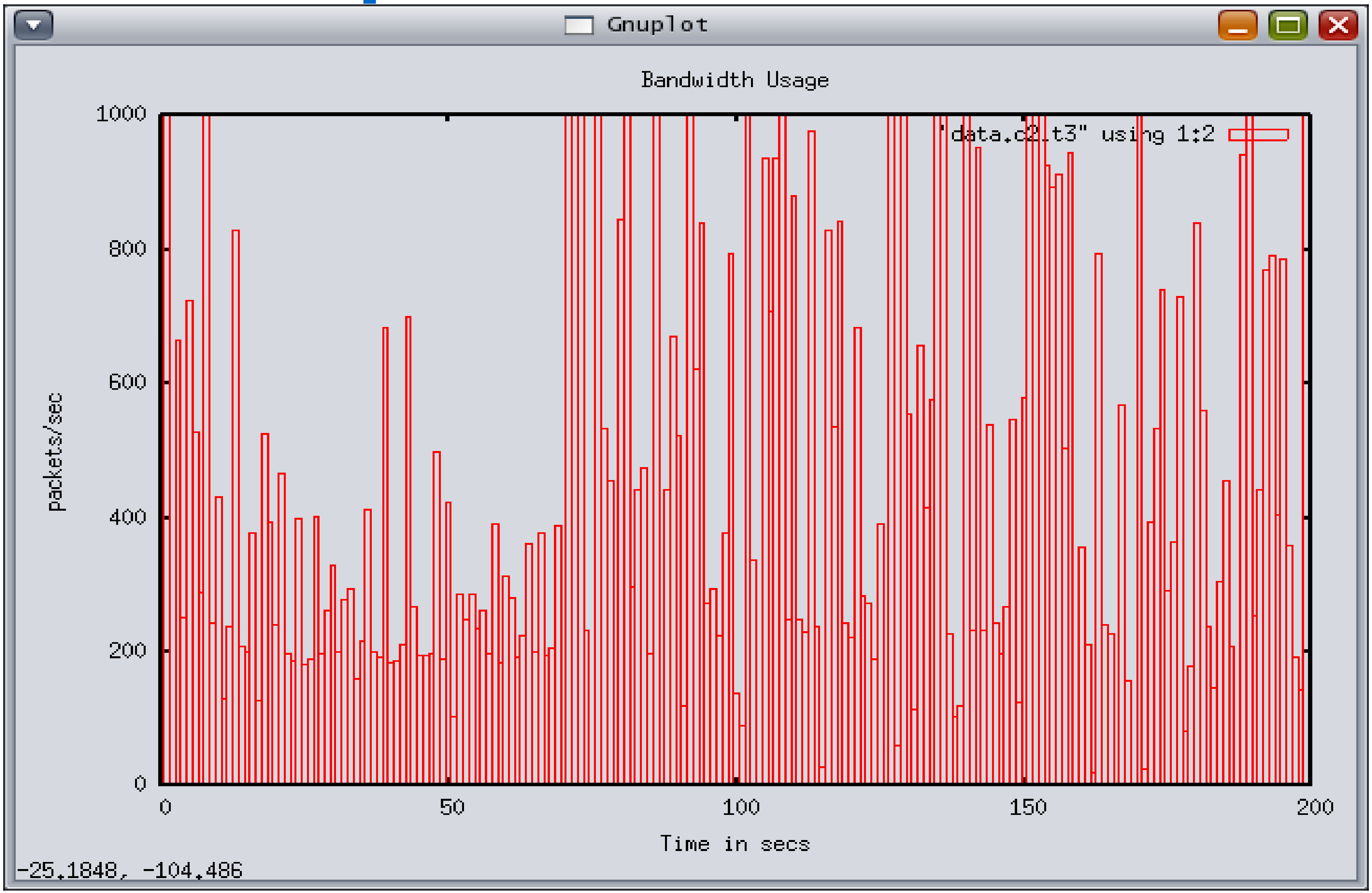
A2D2V2 Full Attack and Response Test Scenario

- Normal tcp_rcv traffic running on C1 and C2, tcp_snd running on S1 and S2 with non-stop TCP SYN flood attack on A1, A2 and A3 targeting both S1 and S2 for 3 ½ minutes. A2D2V2 IDIP enabled Snort running on S1, IDIP firewall/router software running on R97, R99, R102. Class based queueing and other QoS techniques as per A2D2 implementation are applied to firewall/routers.

A2D2V2 Full Attack and Response Results, C1



A2D2V2 Full Attack and Response Results, C2



A2D2V2 Full Attack and Router Response Times

Event	Time
R99 Receives First Attack notification and starts tracing	0
R99 Sends out first attack notification to upstream router R102	T + 6 seconds
R102 Receives first attack notification from R99	T + 9 seconds
R97 Receives first attack notification from R99	T + 62 seconds
R99 Applies first attack rule to itself	T + 65 seconds

A2D2V2 IDIP Communication Between IDIP firewall/routers

idip_firewall_receiver.c do_trace_request: UNDER ATTACK: ***<-- trace request being processed***
idip_firewall_receiver.c do_trace_request: from source 192.168.16.133
idip_firewall_receiver.c do_trace_request: on interface eth3
idip_firewall_receiver.c do_trace_request: number of packets 308
*idip_firewall_receiver.c do_request: message received FLOOD DETECTED on r993 from 192.168.16.133 (THRESHOLD 50 connections exceeded in 10 seconds)****<--creation of IDIP FLOOD message***
idip_firewall_receiver.c do_request: Connected to rate limiter
idip_firewall_receiver.c do_request: Sent msg FLOOD DETECTED on r993 from 192.168.16.133 (THRESHOLD 50 connections exceeded in 10 seconds) to rate limiter
idip_firewall_receiver.c do_trace_request: alertmsg sent to 192.168.14.102: FLOOD DETECTED on r993 from 192.168.16.133 (THRESHOLD 50 connections exceeded in 10 ***<-- alertmsg sent to upstream router, 14.102***
seconds)
idip_firewall_receiver.c do_trace_request : Checking for other upstream routers to notify
idip_firewall_receiver.c do_trace_request(): alertmsg sent to 192.168.12.97: FLOOD DETECTED on r993 from 192.168.16.133 ***<--same message sent to other upstream router, 12.97***

A2D2V2 Cooperative Defense Highlights

- Without cooperative defense of A2D2V2 C2 would starved out during the attack
- Local attack response of A2D2 in place doesn't stop this situation. A2D2V2 provides additional levels of attack detection and response.
- 16.x and 11.x subnets have no attack detection mechanism. Rely on notification from 13.x subnet attack detection to stop attack traffic
- IDS in 13.x had much less work to do since attack was pushed upstream, closer to source
- Multi-administrative domain(A2D2V2 neighborhoods) response is much faster than if human intervention is required

A2D2V2 Conclusions

- Cooperative, multi-network intrusion detection and response system
- A2D2V2 clients on IDIP enabled networks experience reasonable network throughput(packets per second measured for A2D2V2) during the attack
- A2D2V2 clients on Non-IDIP enabled networks experience benefits of IDIP cooperative detection and response in other networks during attack
- Allows victim networks to identify and stop attack at source

Lessons Learned

- So many...
  - How to setup an enterprise network test bed
  - How to setup static routing tables on routers for networks not within 1 link
  - Iptables with multiple input/output interfaces
  - IP forwarding and how it works
  - Linux firewall security
  - Linux
  - Remote management of test bed
  - Hardware setup and configuration
  - Stacheldraht attack tools quirks
  - SSH and X11 forwarding

Future work

- Correlation Engine
- IDIP Enhancements
- Redundant/cooperative discovery coordinators
- OpenSLP
- IDMEF, IDXP, CISL and IDIP
- CIDF
- Performance Enhancements
- Tracing and locating of other IDIP networks

Backup Slides

Pieces of IDIP Implementation for A2D2V2

IDIP Message Header:

```
struct idip_header {
uint18_t   version;
uint8_t    flags;
uint16_t   length;
uint8_t    next_type;
uint8_t    pad;
uint16_t   checksum;
uint32_t   seq_num;
uint32_t   time_stamp;
uint32_t   priority;
uint32_t   dest_addr;
uint32_t   dest_proc_id;
uint32_t   dest_boot_time;
uint32_t   pad_extra;
};
```

Pieces of IDIP Implementation for A2D2V2

IDIP App Header:

```
struct idip_app_msg_hdr {
    uint8_t                     version;
    uint8_t                     class_id;
    uint32_t                    length;
    uint32_t                    timestamp;
    uint32_t                    thread_id;
    struct idip_app_orig_addr   orig_addr;
    uint8_t                     flags;
    uint8_t                     pad[3];
};
```

IDIP vs. IDMEF

- IDMEF defines data formats and exchange procedures for sharing data from IDS system to other IDS systems and to mgt systems interacting with them
- Two open source IDMEF libraries available for IDMEF, libidmef and a Java IDMEF classes
- Both IDMEF and IDIP enable interoperability among opensource commercial and research IDS systems
- IDMEF is XML based, makes it highly interoperable. IDIP uses a message protocol
- IDIP requires additional software infrastructure on IDIP nodes. IDMEF only requires use of the lib/java class to generate the appropriate message.

IDIP vs. IDMEF

- IDIP and IDMEF require knowledgeable party to help correlate data
- IDMEF has some correlation protocol definitions
- IDIP relies on trace message data to determine appropriate responses
- IDMEF is an open, fully available protocol
- IDIP documentation is not fully available. The IDIP Key distribution and Crypotgraphic extensions are not available

IDMEF Model

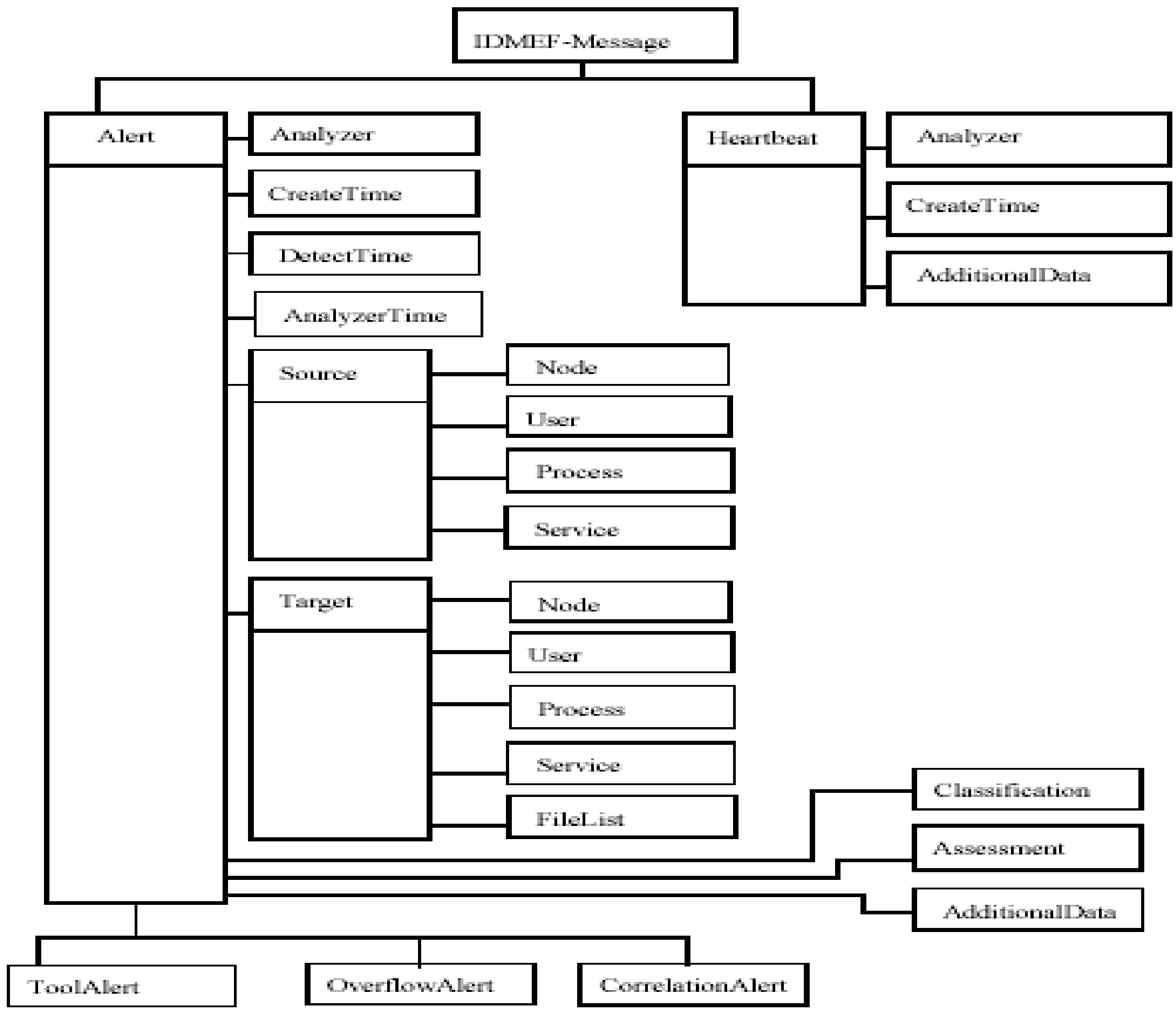


Figure 5. A simplified version of the IDMEF model as of January 30, 2003 [7].

IDIP and CISL

- CISL is IDIP information specification language
- It is used in IDIP to communication trace and report information
- CISL uses S- expression syntax to form sentences describing events and responses
- CISL provides reasonably rich vocabulary for the structure and instances of a set of events involving only networked computers.
- CISL has some limitations

IDIP and CISL

- Example CISL expression for a simple event:

```
Delete
    (When
        (Time '12:24 15 Mar 1999 UTC')
    )
    (Initiator
        (UserName 'joe')
        (UserID 1234)
        (HostName 'foo.example.com')
    )
    (FileSource
        (FullPathName '/etc/passwd')
        (HostName 'foo.example.com')
    )
)
```

IDIP and IDMEF

- CISL seems a bit cumbersome
- Using IDMEF(XML) to transfer data in a compatible way might be more lightweight

IDIP and CIDF

- Effort to develop protocols and application programming interfaces so that IDS research projects can share information and resources to enable sharing of IDS components
- Utilizes CISL for data format
- CIDF's primary goal is to represent intrusion detection data in a Global Intrusion Detection Object(GIDO) format
- Last substantial work done for CIDF in 1999
- CIDF is intended for use in conjunction with IDIP

IDIP vs. IDXP

- IDXP is Intrusion Detection Exchange Protocol used for exchanging data between IDS entities
- Supports mutual authentication, integrity and confidentiality over a connection-oriented protocol
- Specified as a Blocks Extensible Exchange Protocol(BEEP)
- Provides for the exchange of IDMEF messages
- IDXP is an open, published standard
- IDIP protocol spec is only partially available
- Both allow for proxy of intermediate nodes to pass along data
- Both provide for a security protocol. IDIP's security protocol is not available at this time.

A2D2V2 Test Scenarios

1. Normal tcp_rcv traffic running on C1 and C2 and tcp_snd running on S2 with no attack. And, no CBQ applied to firewall/routers. This was used for baseline packet performance data.
2. Normal tcp_rcv traffic running on C1 and C2,tcp_snd running on S1 with the TCP SYN flood attack running on A1, A2 and A3 targeting S1, 192.168.13.1 and S2, 192.168.15.1. No IDIP or IDS software running nor class based queueing has been applied. This is to show the affect on the clients with no DDoS attack mitigation. Results shown are for C1 only. C2 exhibited exact symptoms as C1 in this test scenario, that is the near total loss of packet transmission.

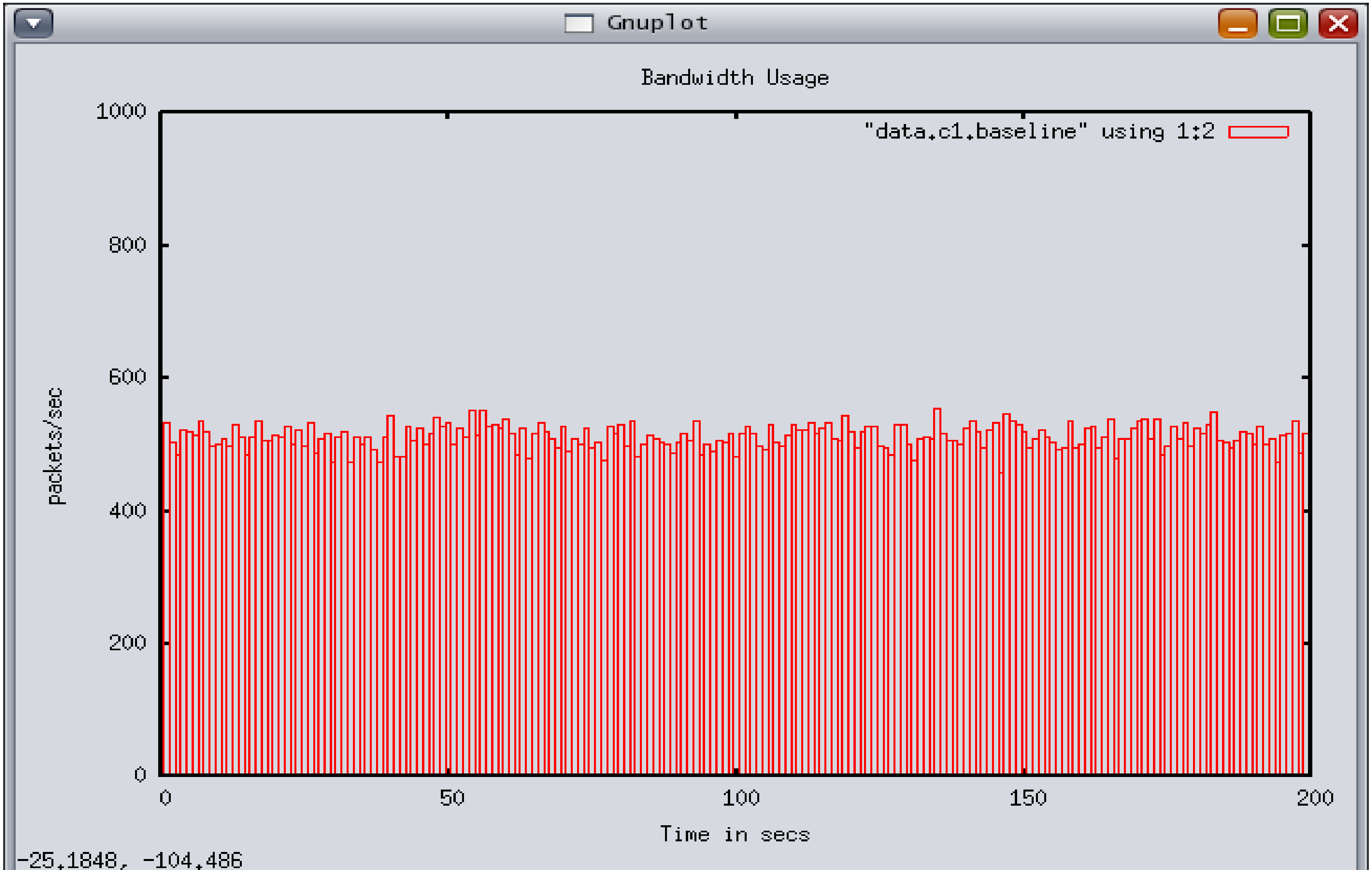
A2D2V2 Test Scenarios

3. Normal tcp_rcv traffic running on C1 and tcp_snd running on S1 with a 3 1/2 minute non-stop TCP SYN attack running on A1 and A2 with R97 and R99 running IDIP enabled software, and S1 running IDIP enabled Snort IDS. Class based queueing and other QoS techniques have been applied to each participating router/firewall as discussed in Section 8.1.2. This scenario is intended to show the attack response within 2 LAN's only. Cooperation happens between the R97 and R99 firewall/routers.

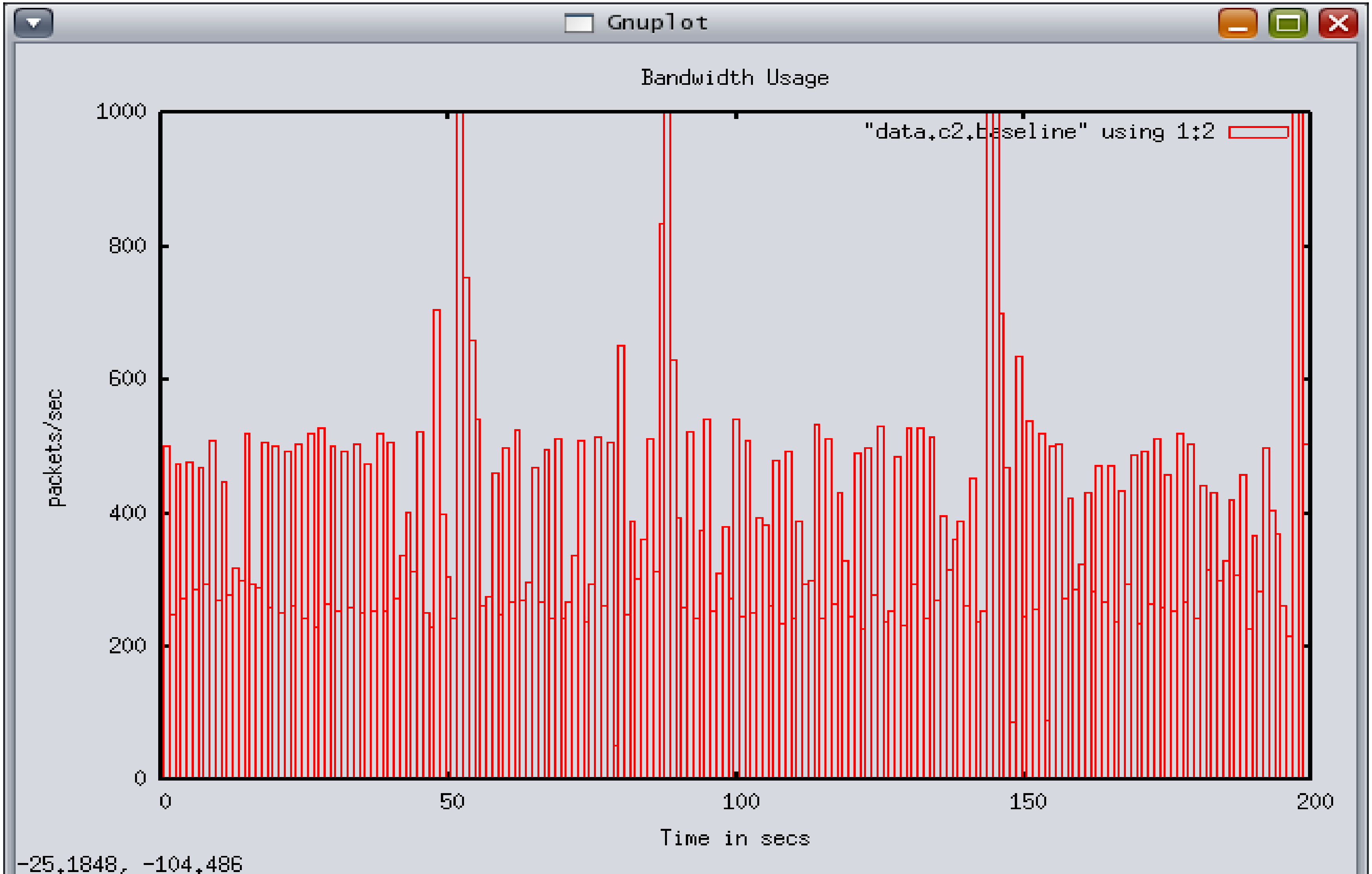
A2D2V2 Test Scenarios

4. Normal tcp_rcv raffic running on C1 and C2, tcp_snd running on S1 and S2 with the non-stop TCP SYN flood attack running on A1, A2 and A3 targeting both S1 and S2 for 3 ½ minutes, along with the A2D2V2 IDIP enabled Snort running on S1, and IDIP firewall/router software running on R97, R99 and R102. Class based queueing and other QoS techniques have been applied to each participating A2D2V2 router/firewall as discussed in Section 8.1.2. This is to show the results of a full enterprise wide cooperative DDoS attack response and mitigation scenario. This test was run several times, with 2 graphs per client being displayed to show the consistency of response for each client.

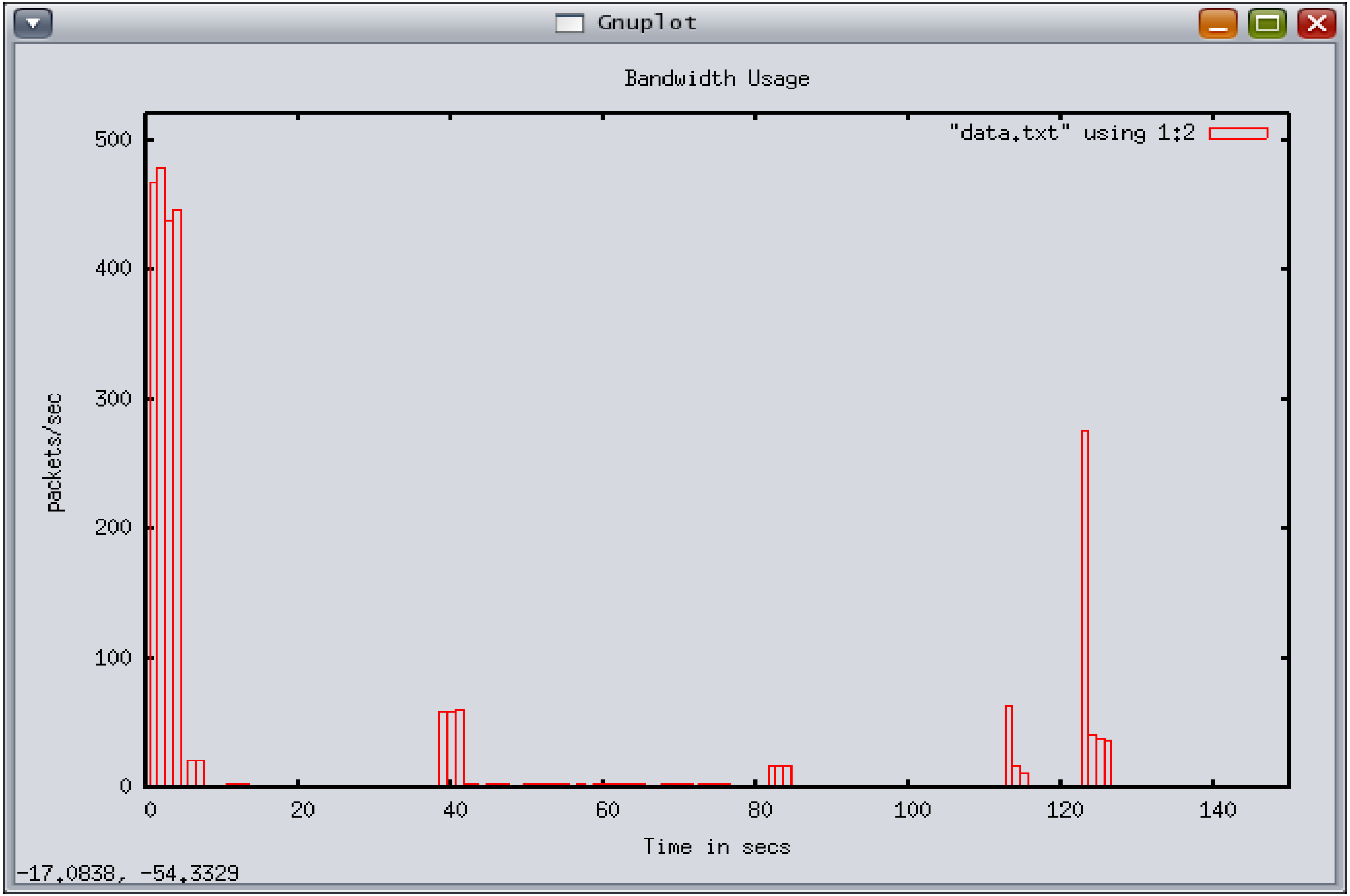
Client C1 Baseline Packet Rate



Client C2 Baseline Packet Rate



Client C1 Test 2 data



Client C1 Test 3 data

A2D2V2 R99 *iptables* After Attack and Mitigation

```
Chain INPUT (policy DROP 25 packets, 3604 bytes)
pkts bytes target     prot opt in     out     source destination
0     0 level3     all  --  any    any     192.168.11.72 anywhere
0     0 level3     all  --  any    any     192.168.11.48 anywhere
0     0 level3     all  --  any    any     192.168.11.114 anywhere
0     0 level3     all  --  any    any     192.168.11.51 anywhere
0     0 level3     all  --  any    any     192.168.11.18 anywhere
0     0 level3     all  --  any    any     192.168.11.134 anywhere
512K  134M ACCEPT     all  --  any    any  anywhere anywhere

Chain FORWARD (policy DROP 0 packets, 0 bytes)
pkts bytes target     prot opt in out     source destination
0     0 level3     all  --  any    any     192.168.11.72  anywhere
0     0 level3     all  --  any    any     192.168.11.48  anywhere
0     0 level3     all  --  any    any     192.168.11.114 anywhere
0     0 level3     all  --  any    any     192.168.11.51  anywhere
0     0 level3     all  --  any    any     192.168.11.18  anywhere
0     0 level3     all  --  any    any     192.168.11.134 anywhere
894K  170M ACCEPT     all  --  any    any     anywhere anywhere

Chain OUTPUT (policy DROP 1 packets, 52 bytes)
pkts bytes target     prot opt in     out     source  destination
286K  102M ACCEPT     all  --  any    any     anywhere anywhere
```

A2D2V2 R99 *iptables* After Attack and Mitigation

```
Chain level0 (0 references)
pkts bytes target     prot opt in     out     source anywhere
0     0 DROP       all  --  any    any     anywhere  anywhere

Chain level1 (0 references)
pkts bytes target     prot opt in     out     source destination
0     0 DROP       all  --  any    any     anywhere anywhere

Chain level2 (0 references)
pkts bytes target     prot opt in     out     source destination
0     0 ACCEPT     all  --  any    any     anywhere anywhere
        limit: avg 50/sec burst 5
0     0 DROP       all  --  any    any     anywhere anywhere

Chain level3 (14 references)
pkts bytes target     prot opt in     out     source anywhere 0     0 ACCEPT     all  --  any    any
      anywhere anywhere
      limit: avg 151/sec burst 5
0     0 DROP       all  --  any    any     anywhere anywhere
```

A2D2V2 R102 *iptables* After Attack and Mitigation

Chain INPUT (policy DROP 0 packets, 0 bytes)

pkts	bytes	target	prot	opt	in	out	source	destination
0	0	level3	all	--	any	any	192.168.11.72	anywhere
0	0	level3	all	--	any	any	192.168.11.48	anywhere
0	0	level3	all	--	any	any	192.168.11.114	anywhere
0	0	level3	all	--	any	any	192.168.11.51	anywhere
0	0	level3	all	--	any	any	192.168.11.18	anywhere
0	0	level3	all	--	any	any	192.168.11.134	anywhere
3544	450K	ACCEPT	all	--	any	any	anywhere	anywhere

Chain FORWARD (policy DROP 0 packets, 0 bytes)

pkts	bytes	target	prot	opt	in	out	source	destination
0	0	level3	all	--	any	any	192.168.11.72	anywhere
0	0	level3	all	--	any	any	192.168.11.48	anywhere
0	0	level3	all	--	any	any	192.168.11.114	anywhere
0	0	level3	all	--	any	any	192.168.11.51	anywhere
0	0	level3	all	--	any	any	192.168.11.18	anywhere
0	0	level3	all	--	any	any	192.168.11.134	anywhere
1799K	253M	ACCEPT	all	--	any	any	anywhere	anywhere

Chain OUTPUT (policy DROP 0 packets, 0 bytes)

pkts	bytes	target	prot	opt	in	out	source	destination
3487	363K	ACCEPT	all	--	any	any	anywhere	anywhere

A2D2V2 R102 *iptables* After Attack and Mitigation

```
 Chain level0 (0 references)
pkts bytes target     prot opt in     out   source destination
0     0 DROP       all  --  any    any     anywhere   anywhere

Chain level1 (0 references)
pkts bytes target     prot opt in     out  source  destination
0     0 DROP       all  --  any    any     anywhere anywhere

Chain level2 (0 references)
pkts bytes target     prot opt in     out   source destination
0     0 ACCEPT     all  --  any    any     anywhere   anywhere
        limit: avg 50/sec burst 5
0     0 DROP       all  --  any    any     anywhere    anywhere

Chain level3 (14 references)
pkts bytes target     prot opt in     out     source
destination
1243 1861K ACCEPT     all  --  any    any     anywhere anywhere
      limit: avg 151/sec burst 5
500  749K DROP
```

A2D2V2 *iptraf* Data From S2 During Attack Run

Wed Jul 5 14:13:05 2006; ******** Detailed interface statistics started ********

*** Detailed statistics for interface eth0, generated Wed Jul 5 14:18:52 2006

Total: 1565701 packets, 210432861 bytes
(incoming: 716189 packets, 45786214 bytes; outgoing: 849512 packets, 164646647 bytes)
IP: 1565701 packets, 186996595 bytes
(incoming: 716189 packets, 34243116 bytes; outgoing: 849512 packets, 152753479 bytes)
TCP: 1565433 packets, 186978371 bytes
(incoming: 715921 packets, 34224892 bytes; outgoing: 849512 packets, 152753479 bytes)
UDP: 0 packets, 0 bytes
(incoming: 0 packets, 0 bytes; outgoing: 0 packets, 0 bytes)
ICMP: 268 packets, 18224 bytes
(incoming: 268 packets, 18224 bytes; outgoing: 0 packets, 0 bytes)
Other IP: 0 packets, 0 bytes
(incoming: 0 packets, 0 bytes; outgoing: 0 packets, 0 bytes)
Non-IP: 0 packets, 0 bytes
(incoming: 0 packets, 0 bytes; outgoing: 0 packets, 0 bytes)
Broadcast: 0 packets, 0 bytes

A2D2V2 *iptraf* Data From S2 During Attack Run

```
Average rates:
  Total:      4851.48 kbits/s, 4512.11 packets/s
  Incoming:   1055.59 kbits/s, 2063.95 packets/s
  Outgoing:   3795.89 kbits/s, 2448.16 packets/s

Peak total activity: 7028.49 kbits/s, 8184.80 packets/s
Peak incoming rate: 2118.14 kbits/s, 4075.20 packets/s
Peak outgoing rate: 5706.25 kbits/s, 4901.00 packets/s
IP checksum errors: 0

Running time: 347 seconds
Wed Jul  5 14:18:52 2006; ******** Detailed interface statistics
   stopped
```

A2D2V2 idip_firewall_receiver main()

```
/*
 * The backplane listens on a socket and determines the type of request
 * being sent to it. From there it invokes the appropriate processing.
 */

void
main() {

        int                         length;
        int                         n;
        idip_message_t              i_message;
        struct sockaddr_in          toaddr;

        /* Set up our listening socket */
        if ((gen_mbx = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
                fprintf(stderr, "Unable to set up receiver socket.\n");
                perror(strerror(errno));
                return;
        } * Listen for messages from any host, on the IDIP_APP_PORT
         */
        (void) memset(&gen_from, 0, sizeof (gen_from));
        gen_from.sin_family = AF_INET;
        gen_from.sin_addr.s_addr = INADDR_ANY;
```

A2D2V2 idip_firewall_receiver main()

```
Gen_from.sin_port = htons(IDIP_APP_PORT);

if (bind(gen_mbx, (struct sockaddr *) &gen_from,
        sizeof (struct sockaddr_in)) < 0) {
        fprintf(stderr, "%s", "Could not bind to port\n");
        perror(strerror(errno));
}

length = sizeof (gen_from);

if (getsockname(gen_mbx, (struct sockaddr *) &gen_from,
    &length)) {
        perror("getting socket name");
        exit(1);
}
while (1) {
        n = recvfrom(gen_mbx, &i_message,
            sizeof (idip_message_t),
            0, (struct sockaddr *)&gen_from, &length);
        if (n < 0) {
            perror("receiving datagram messages");
            continue;
        }
```

A2D2V2 idip_firewall_receiver main()

```
                /*
                 * Process this message. It is possible that there has
                 * been a transmission problem or data is garbled.
                 *  Move on
                 * if this is the case.
                 */
                if (process_idip_message(&i_message) != 0) {
                        perror("error processing idip message");
                        continue;
                }
        }

        /*NOTREACHED*/
}
```

A2D2V2 tcpdump.sh

```
# set time limit based on what caller specified. Exec script that will send
# SIGTERM to tcpdump to force this script to run the END block. Background
# this so it doesn't interrupt gawk processing below.

# Invoke tcpdump with options and pipe through gawk to gather data. The
# running of tcpdump is limited to the time specified by the caller. I
# am only interested in the ip protocol packets. I will get the source
# and destination addresses with the ''ip' specifier at $3 and $5 respectively.
# Do not track outgoing packets from this host as part of tracing data. This is
# achieved by the 'src host not loghost' qualifier.

#
# I need to dump on every interface I find on system. so, call ifconfig -a
# first, to get interface name. Call tcpdump on these.

INTERFACES=`/sbin/ifconfig | gawk ' {
        # Get the interface name
        x = split($1, ifname)
        newif[i]=ifname[1]
        if (match(newif[i], "eth") && newif[i] != "lo") {
             printf("%s ", newif[i])
        }
```

A2D2V2 tcpdump.sh

```
# I need to dump on every interface I find on system. so, call ifconfig -a
# first, to get interface name. Call tcpdump on these.

INTERFACES=`/sbin/ifconfig | gawk ' {
        # Get the interface name
        x = split($1, ifname)
        newif[i]=ifname[1]
        if (match(newif[i], "eth") && newif[i] != "lo") {
                printf("%s ", newif[i])
        }
        i = i + 1
} '`
for i in $INTERFACES
do
# for each interface check number of packets , if over threshold, report
./dumper.sh $i $1 > /tmp/o_$i &
done
# kill this process in $1 amount of time
./trace_kill $2
sleep 3
/bin/cat /tmp/o_*
```