ภาคผนวก ก วิธีการติดตั้งโปรแกรม NS 2.0 บนระบบปฏิบัติการลีนุกซ์ RedHat 6.2

1. ทำการดาวน์โหลดโปรแกรมซึ่งเป็น Zip ไฟล์ชื่อ ns-allinone-2.1b8a.tar.gz ขนาด 49.0 Mbytes ที่

http://www.isi.edu/nsnam/ns/ns-build.html

2. ทำการ Unzip ไฟล์ที่ได้ดาวน์โหลดมาคือ ns-allinone-2.1b8a.tar.gz ด้วยคำสั่งดังนี้

tar –zxvf ns-allinone-2.1b8a.tar.gz

จะได้โฟล์เดอร์ชื่อ ns-allinone-2.1b8a

3. ทำการติดตั้งโปรแกรมซึ่งอยู่ใน ns-allinone-2.1b8a ด้วยคำสั่งดังนี้

./install

4. ตรวจสอบความถูกต้องของการติดตั้ง ซึ่งอยู่ใน /ns-allinone-2.1b8a/ns-2.1b8a/ ด้วยคำสั่งดังนี้

./validate

เป็นอันเสร็จสิ้นการติดตั้งโปรแกรม NS 2.0

ภาคผนวก ข ไฟล์ที่ได้แก้ไขและเพิ่มเติม

ไฟล์ชื่อ ns-lib.tcl

```
840a841,845
>           ##added for wfq
>           WFQ {
>               set q [new Queue/$qtype]
>               $q set bandwidth_ $bw
>           }
1265a1271,1276
> }
>
> ##added for wfq
> Simulator instproc wfqclassifier-install { n1 n2 clssf } {
>       $self instvar link_
>       [$link_([$n1 id]:[$n2 id]) queue] install $clssf
```

ไฟล์ชื่อ ns-default.tcl

```
88a89,90
> Queue/WFQ set bandwidth_ 0
>
92a95,98
> ##added for prio
> Queue/Prio set prio_queue_ true
> Queue/Prio set drop_front_ false
>
669a676,678
> ##added for tcpsink
> Agent/TCPSink set bytes_        0
>
```

ไฟล์ชื่อ Tcp-sink.cc

```
152,159c152,155
<       /*
<        * maxSackBlocks_ does wierd tracing things.
<        * don't make it delay-bound yet.
<        */
< #if defined(TCP_DELAY_BIND_ALL) && 0
< #else /* ! TCP_DELAY_BIND_ALL */
<       bind("maxSackBlocks_", &max_sack_blocks_); // used only by sack
< #endif /* TCP_DELAY_BIND_ALL */
---
>       bind("packetSize_", &size_);  /*added*/
>       bind("maxSackBlocks_", &max_sack_blocks_); /* used only by sack*/
>       bind_bool("ts_echo_bugfix_", &ts_echo_bugfix_);  /*added*/
>       bind("bytes_", &bytes_); bytes_=0;  /*added*/
161a158
>
294c291,292
<       if (numToDeliver)
---
>       if (numToDeliver) {
>               bytes_+=numToDeliver;   /*added*/
295a294
>       }
```

ไฟล์ชื่อ Tcp-sink.h

```
100a101
>       int bytes_;   /*added*/
```

ไฟล์ชื่อ prio.h

```
#ifndef ns_prio_h
#define ns_prio_h

#include <string.h>
#include "queue.h"
#include "packet.h"
#include "config.h"

/* A bounded, prio queue */
class Prio : public Queue {
 public:
        Prio() {
                q1_ = new PacketQueue;
                q2_ = new PacketQueue;
                q3_ = new PacketQueue;
                q4_ = new PacketQueue;
                bind_bool("prio_queue_", &prio_queue_);
                bind_bool("drop_front_", &drop_front_);
        }

        ~Prio() {
                delete q1_;
                delete q2_;
                delete q3_;
                delete q4_;
        }

        void recvOther(Packet* p);
        /* insert packet at front of queue */
        void recvHighPrio(Packet*);
        void recvMedPrio(Packet*);
        void recvNormPrio(Packet*);
        void checkqlen();
```

```
 protected:
        Packet* deque();
        void enque(Packet*);
        PacketQueue *q1_;       /* underlying FIFO queue */
        PacketQueue *q2_;
        PacketQueue *q3_;
        PacketQueue *q4_;
        int prio_queue_;
        int drop_front_;
};

#endif /*!_prio_h*/
```

ไฟล์ชื่อ prio.cc

```
#include "prio.h"
#include <packet.h>
#include <queue.h>
#include <stdio.h>

static class PrioClass : public TclClass {
 public:
        PrioClass() : TclClass("Queue/Prio") {}
        TclObject* create(int, const char*const*) {
                return (new Prio);
        }
} class_prio;

void Prio::enque(Packet* p)
{
        if (prio_queue_) {
                hdr_ip* hd = hdr_ip::access(p);
                switch (hd->prio()) {
                        case 0 :
                                recvHighPrio(p);
                                break;
                        case 1 :
                                recvNormPrio(p);
                                break;
                        case 2 :
                                recvMedPrio(p);
                                break;
                        default :
                                recvOther(p);
                }
        }
}

Packet* Prio::deque()
```

```
{
        if (q1_->length() != 0) {
                return q1_->deque();
        }
        else {
                if (q2_->length() != 0) {
                        return q2_->deque();
                }
                else {
                        if (q3_->length() != 0) {
                                return q3_->deque();
                        }
                        else
                                return q4_->deque();
                }
        }
}

void Prio::recvHighPrio(Packet* p)
{
        // prio 0
        q1_->enque(p);
        checkqlen();
}

void Prio::recvNormPrio(Packet* p)
{
        // prio 1
        q2_->enque(p);
        checkqlen();
}

void Prio::recvMedPrio(Packet* p)
{
        //prio 2
```

```
        q3_->enque(p);
        checkqlen();
}

void Prio::recvOther(Packet* p)
{
        //prio 3
        q4_->enque(p);
        checkqlen();
}

void Prio::checkqlen() {
        if (q1_->length() + q2_->length() + q3_->length() + q4_->length() >= qlim_) {
                /* remove from head of queue */
                if (drop_front_) {
                        // have data in q4, drop in q4
                        if (q4_->length() != 0) {
                                Packet* drop_q4 = q4_->deque();
                                drop(drop_q4);
                        }

                        else {
                                // no have data in q4, drop in q3
                                if (q3_->length() != 0) {
                                        Packet* drop_q3 = q3_->deque();
                                        drop(drop_q3);                          }
                                // no have data in q3, drop in q2
                                else {
                                        if (q2_->length() != 0) {
                                                Packet* drop_q2 = q2_->deque();
                                                drop(drop_q2);
                                        }
                                        else {
                                                Packet* drop_q1 = q1_->deque();
                                                drop(drop_q1);
```

```
                        }
                    }
                }
            }
            else {
                // have data in q4, drop in q4
                if (q4_->length() != 0) {
                    Packet* drop_q4 = q4_->lookup(q4_->length()-1);
                    q4_->remove(drop_q4);
                    drop(drop_q4);
                }
                else {
                    // no have data in q4, drop in q3
                    if (q3_->length() != 0) {
                        Packet* drop_q3 = q3_->lookup(q3_->length()-1);
                        q3_->remove(drop_q3);
                        drop(drop_q3);
                    }
                    // no have data in q3, drop in q2
                    else {
                        if (q2_->length() != 0) {
                            Packet* drop_q2 = q2_->lookup(q2_->length()-1);
                            q2_->remove(drop_q2);
                            drop(drop_q2);
                        }
                        else {
                            Packet* drop_q1 = q1_->lookup(q1_->length()-1);
                            q1_->remove(drop_q1);
                            drop(drop_q1);
                        }
                    }
                }
            }
        }
    }
```

ไฟล์ชื่อ wfqsamplec.cc

```
#include "wfqclassifier.h"

class WFQSampleClassifier : public WFQClassifier {
        public:
                WFQSampleClassifier();
                virtual int command(int argc, const char*const* argv);
                int get_queueid(Packet *p);
                double get_weight(int queueid);
        protected:
                double private_weight[MAXQUEUE];
};

static class WFQSampleClassifierClass : public TclClass {
        public:
                WFQSampleClassifierClass():TclClass("WFQSampleClassifier") {};
                TclObject* create(int,const char*const*) {
                        return(new WFQSampleClassifier);
                };
} class_wfqsampleclassifier;

WFQSampleClassifier::WFQSampleClassifier() {
        min_weight=10e+10;
        for(int i=0;i<MAXQUEUE;i++)
                private_weight[i]=1;
}

int WFQSampleClassifier::command(int argc,const char*const* argv) {
        if (argc == 4) {
                if(!strcmp(argv[1],"setweight"))  {
                        double weight;
                        int flow,success=0;
                        success += sscanf(argv[2],"%d",&flow);
                        success += sscanf(argv[3],"%lf",&weight);
                        if(success==2)  {
```

```
                        if ( !(flow<MAXQUEUE) ) {
                                fprintf(stderr,
                                "flow id is out of range: edit MAXFLOW in wfqclassifier.h and
                                recompile\n");
                                exit(1);
                        }
                        min_weight = min_weight<weight ?min_weight:weight;
                        private_weight[flow]=weight;
                        return TCL_OK;
                }
        }
    }
    return (TclObject::command(argc, argv));
}

inline int WFQSampleClassifier::get_queueid(Packet *p) {
    hdr_ip* h = hdr_ip::access(p);
    return h->flowid();
}

inline double WFQSampleClassifier::get_weight(int queueid) {
    return private_weight[queueid];
}
```

ไฟล์ชื่อ wfq-list.h

```
#ifndef LIST_H
#define LIST_H

template <class X> struct elem {
        double key;
        X data;
        elem *next;
        elem *prev;
        elem(double k=0,X d=0,elem *n=0,elem *p=0) : key(k),data(d),next(n),prev(p) {};
};

template <class X> class List {
        protected:
                elem<X> *head,*tail;
        public:
                List(): head(0), tail(0) {};

                void insert_order(X el,double k,int flowID) {
                        elem<X> *tmp=new elem<X>(k,el);
                        if(head==0 && tail==0) { //  list is empty
                                tail = head = tmp;
                        } else if (k >= tail->key ) { // insert as first
                                        tmp->next=tail;
                                        tail->prev=tmp;
                                        tail = tmp;
                        } else {
                                elem<X> *p=tail; // other cases
                                while((p->next)!=0) {
                                        if( (p->next)->key <= k ) break;
                                        p=p->next;
                                }
                                if(p->next != 0) {
                                        p->next->prev = tmp;
                                        tmp->next = p->next;
```

```
                } else
                        head=tmp;
                p->next=tmp;
                tmp->prev=p;
        }
};

double get_key_min() {
        if(head !=0)
                return head->key;
        else
                return -1;
};

X get_data_min() {
        if(head != 0)
                return head->data;
        else
                return 0;
};

int extract() {
        if(head != 0) {
                if(head->prev != 0) {
                        elem<X> *new_head=head->prev;
                        new_head->next=0;
                        delete head;
                        head=new_head;
                } else {
                        delete head;
                        head=0;
                        tail=0;
                }
                return 0;
        }
```

```
                return 1;
        };
};
#endif
```

ไฟล์ชื่อ wfqclassifier.h

```
#ifndef WFQ_CLASSIFIER_H
#define WFQ_CLASSIFIER_H

#include "packet.h"
#include "queue.h"

#define MAXQUEUE 1024

class WFQClassifier : public TclObject {
        public:
                virtual int get_queueid(Packet *p)=0;
                virtual double get_weight(int queueid)=0;
                virtual unsigned int get_length(int queueid) {return 10; }
                virtual double get_safe_limit() { return min_weight/1000; }
        protected:
                double min_weight;
};

class WFQDefaultClassifier : public WFQClassifier {
        public:
                WFQDefaultClassifier();
                int get_queueid(Packet *p);
                double get_weight(int queueid);
};

inline WFQDefaultClassifier::WFQDefaultClassifier() {
        min_weight=1;
}

inline int WFQDefaultClassifier::get_queueid(Packet *p) {
        hdr_ip* h = hdr_ip::access(p);
        return h->flowid();
}
```

```
inline double WFQDefaultClassifier::get_weight(int queueid) {
        return 1;
}

#endif
```

ไฟล์ชื่อ wfqaggreg.cc

```
#include "wfqclassifier.h"
#define MAXFLOW 1024
#define wfqerror(n) fprintf(stderr,n "\nedit MAXFLOW and MAXQUEUE in wfqaggreg.cc and
wfqclassifier.h, then recompile.\n"); exit(1)

class WFQAggregClassifier : public WFQClassifier {
        public:
                WFQAggregClassifier();
                virtual int command(int argc, const char*const* argv);
                int get_queueid(Packet *p);
                double get_weight(int queueid);
                unsigned int get_length(int queueid);
        protected:
                double private_weight[MAXQUEUE];
                unsigned int private_length[MAXQUEUE];
                int f2q[MAXFLOW];
};

static class WFQAggregClassifierClass : public TclClass {
        public:
                WFQAggregClassifierClass():TclClass("WFQAggregClassifier") {};
                TclObject* create(int,const char*const*) {
                        return(new WFQAggregClassifier);
                };
} class_wfqsampleclassifier;

WFQAggregClassifier::WFQAggregClassifier() {
        min_weight=10e+10;
        for(int i=0;i<MAXFLOW;i++) {
                f2q[i]=-1;
        }
        for(int i=0;i<MAXQUEUE;i++) {
                private_weight[i]=1;
                private_length[i]=10;
```

```
        }
}

int WFQAggregClassifier::command(int argc,const char*const* argv) {
        if (argc == 4) {
                if(!strcmp(argv[1],"setweight"))  {
                        double weight;
                        int queue,success=0;
                        success += sscanf(argv[2],"%d",&queue);
                        success += sscanf(argv[3],"%lf",&weight);
                        if(success==2)  {
                                if ( !(queue<MAXQUEUE) ) {
                                        wfqerror("queue id out of range: ");
                                }
                                min_weight= min_weight<weight ?
                                min_weight:weight;
                                private_weight[queue]=weight;
                                return TCL_OK;
                        }
                }
                else if(!strcmp(argv[1],"setqueue")) {
                        int queue,flow,success=0;
                        success += sscanf(argv[2],"%d",&flow);
                        success += sscanf(argv[3],"%d",&queue);
                        if(success==2) {
                                if ( !(queue<MAXQUEUE) ) {
                                        wfqerror("queue id out of range: ");
                  }
                                if ( !(flow<MAXFLOW) ) {
                                        wfqerror("flow id out of range: ");
                  }
                                f2q[flow]=queue;
                                return TCL_OK;
                        }
                }
```

```
                else if(!strcmp(argv[1],"setlength")) {
                        int queue,length,success=0;
                        success += sscanf(argv[2],"%d",&queue);
                        success += sscanf(argv[3],"%d",&length);
                        if(success==2) {
                                private_length[queue]= length >=0 ? length : 0;
                                return TCL_OK;
                        }
                }
        }
        return (TclObject::command(argc, argv));
}

inline int WFQAggregClassifier::get_queueid(Packet *p) {
        hdr_ip* h = hdr_ip::access(p);
        int queue=f2q[h->flowid()];
        if (queue == -1) {
                abort();
        }
        return queue;
}

inline double WFQAggregClassifier::get_weight(int queue) {
        return private_weight[queue];
}

inline unsigned int WFQAggregClassifier::get_length(int queue) {
        return private_length[queue];
}
```

ไฟล์ชื่อ wfq.cc

```
#include "wfq-list.h"
#include "math.h"
#include "queue.h"
#include "wfqclassifier.h"

#ifndef MAXQUEUE
#warning "MAXQUEUE should be defined in wfqclassifier.h"
#define MAXQUEUE 32
#endif

class WFQ;

class WFQHandler : public Handler {
        public:
                WFQHandler(WFQ *queue) : q(queue){};
                void handle(Event *e);
        protected:
                WFQ* q;
};

class WFQ : public Queue {
        public:
                WFQ();
                virtual int command(int argc, const char*const* argv);
                Packet *deque(void);
                void enque(Packet* pkt);
                void handle(Event *);
        protected:
                void scheduleWFQ();
                WFQHandler wfq_hand;
                Event *wfq_event;
                int idle;
                double virt_time;
                double last_vt_update;
```

```
        unsigned int B[MAXQUEUE];
        unsigned int npacket[MAXQUEUE];
        double finish_t[MAXQUEUE];
        double sum;
        WFQClassifier *wfq_classifier;
        List<int> GPS_queueID_l;
        List<Packet *> PGPS_pack_l;
        double bandwidth;
};

static class WFQClass : public TclClass {
    public:
        WFQClass() : TclClass("Queue/WFQ") {};
        TclObject* create(int, const char*const*) {
            return (new WFQ);
        }
} class_wfq;

WFQ::WFQ() :wfq_hand(this) {
    wfq_classifier=(WFQClassifier*) new WFQDefaultClassifier();
    for(int i = 0;i < MAXQUEUE;i++) {
        npacket[i]=finish_t[i]=B[i]=0;
    }
    wfq_event=0;
    virt_time=sum=0;
    last_vt_update=0;
    idle=1;
    bind_bw("bandwidth_",&bandwidth);
}

int WFQ::command(int argc,const char*const* argv) {
    if (argc==3) {
        if(strcmp(argv[1],"install")==0) {
            wfq_classifier=(WFQClassifier *)TclObject::lookup(argv[2]);
            return TCL_OK;
```

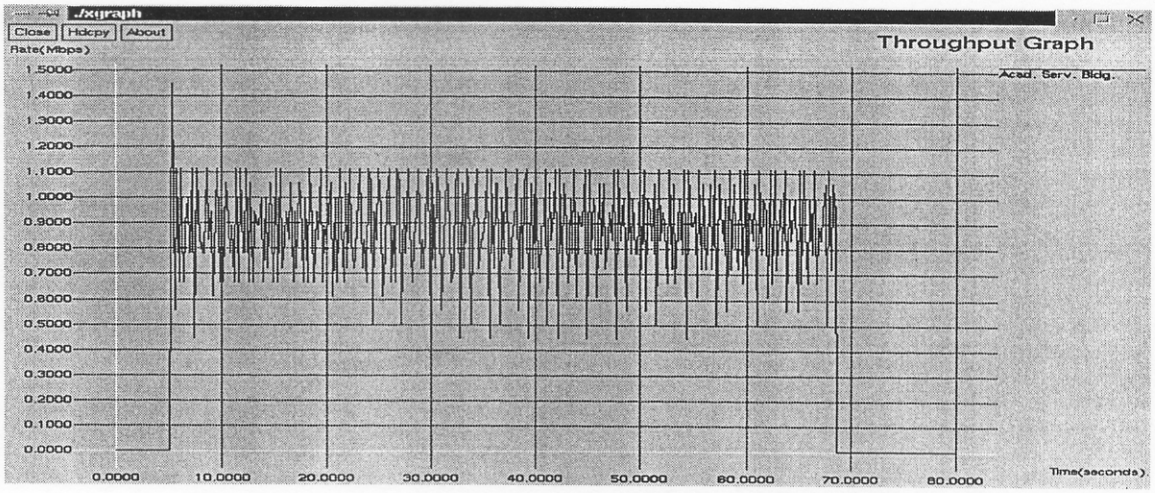
```
                }
        }
        return(Queue::command(argc,argv));
}

void WFQ::enque(Packet *p) {
        int queueid = wfq_classifier->get_queueid(p);
        if (wfq_classifier->get_length(queueid) < npacket[queueid]+1 ) {
                drop(p);
                return;
        }
        npacket[queueid]++;
        hdr_cmn *hdr = hdr_cmn::access(p);
        int size = hdr->size();
        double now =  Scheduler::instance().clock();

        if(idle) {
                last_vt_update=now;
                virt_time=0;
                idle=0;
        } else {
                virt_time=virt_time+(now-last_vt_update)/sum;
                last_vt_update=now;
        }

        finish_t[queueid] = (finish_t[queueid] > virt_time ?
                        finish_t[queueid]:virt_time)
                        +size/wfq_classifier->get_weight(queueid)/(bandwidth/8);

        if((B[queueid]++)==0)
                sum=sum+wfq_classifier->get_weight(queueid);
        if ( fabs(sum) < wfq_classifier->get_safe_limit() ) sum=0;

        PGPS_pack_l.insert_order(p,finish_t[queueid],queueid);
        GPS_queueID_l.insert_order(queueid,finish_t[queueid],queueid);
```

```
        if(wfq_event!=0) {
                Scheduler::instance().cancel(wfq_event);
                delete wfq_event;
        }
        scheduleWFQ();
}

void WFQ::scheduleWFQ() {
        wfq_event=new Event();
        double tmp=(GPS_queueID_l.get_key_min()-virt_time)*sum;
        if (tmp<0) tmp=0;
        Scheduler::instance().schedule((Handler *)&wfq_hand,wfq_event,tmp);
}

void WFQ::handle(Event *e) {
        double now = Scheduler::instance().clock();
        virt_time=virt_time+(now-last_vt_update)/sum;
        last_vt_update=now;
        int queueid=GPS_queueID_l.get_data_min();
        GPS_queueID_l.extract();

        if((--B[queueid])==0)
                sum=sum-wfq_classifier->get_weight(queueid);
        if ( fabs(sum) < wfq_classifier->get_safe_limit() ) sum=0;
                if(sum==0) {
                idle=1;
                for(int i=0;i < MAXQUEUE;i++) finish_t[i]=0;
        }
        delete e;
        if(!idle)
                scheduleWFQ();
        else
                wfq_event=0;
}
```

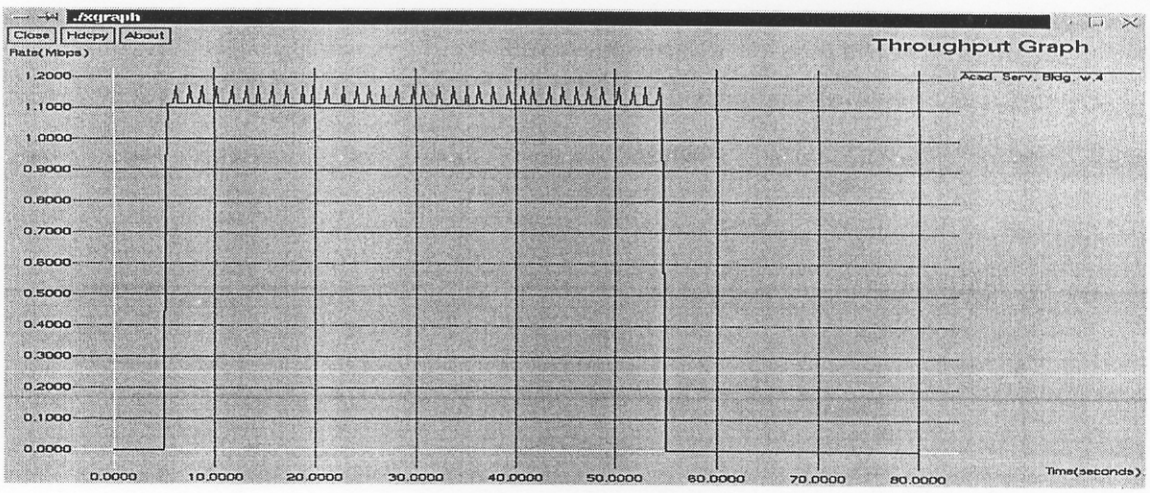
```
Packet* WFQ::deque() {
        Packet* p=PGPS_pack_l.get_data_min();
        PGPS_pack_l.extract();
        if (p) npacket[wfq_classifier->get_queueid(p)]--;
        return p;
}

inline void WFQHandler::handle(Event* e) { q->handle(e); }
```

ภาคผนวก ค ผลการทดสอบของแต่ละคณะ/หน่วยงานก่อนและหลังการจัดคิว จากข้อมูลสมมุติ

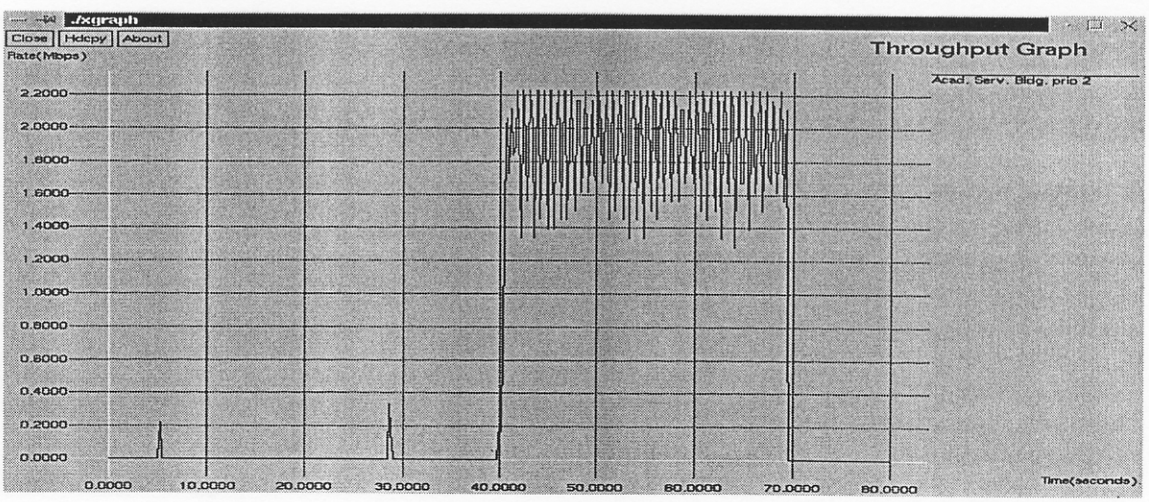
(ก) ก่อนการจัดคิวของอาคารบริหารวิชาการรวม



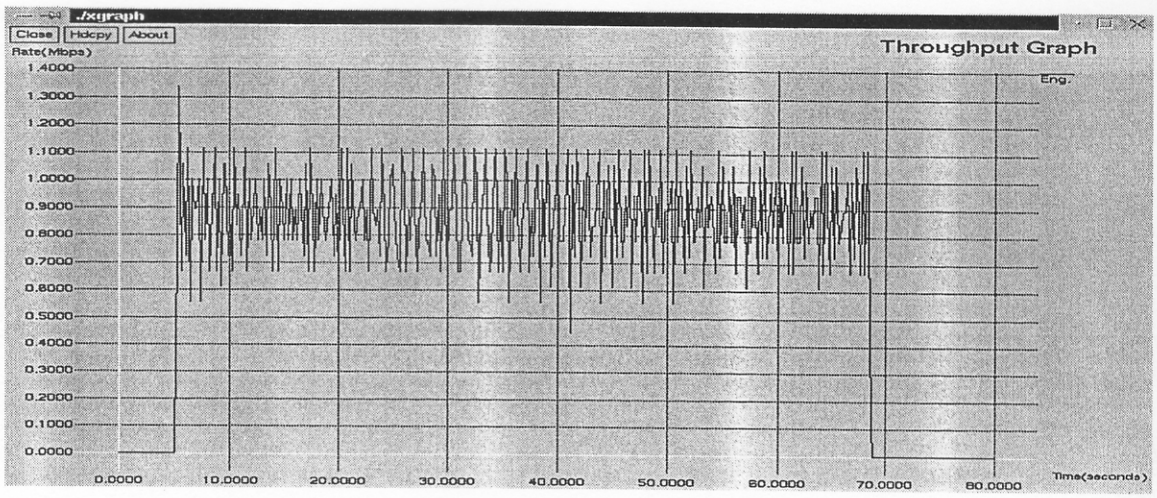
(ข) หลังการจัดคิวแบบ WFQ



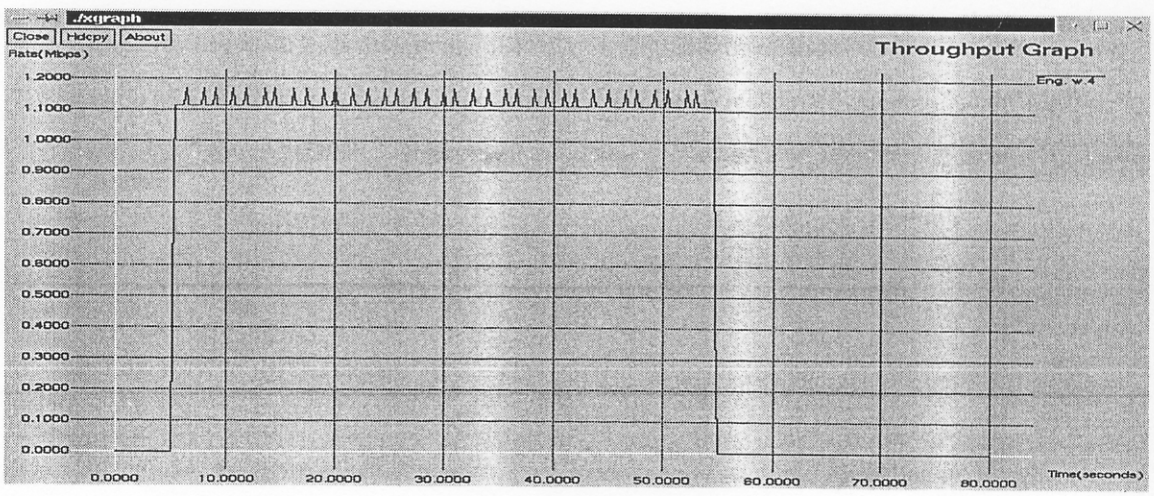
(ค) หลังการจัดคิวแบบ PQ



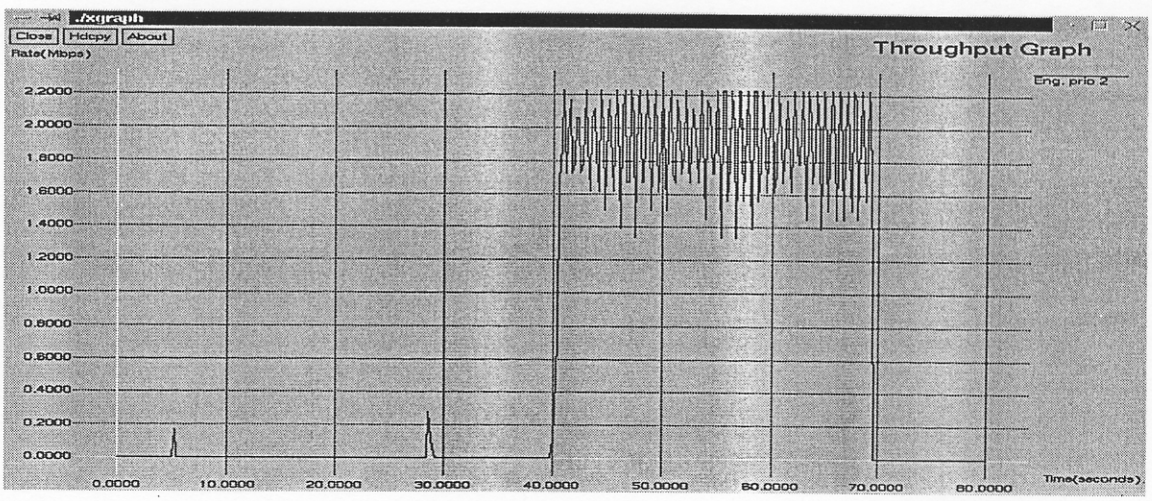
(ก) ก่อนการจัดคิวของคณะวิศวกรรมศาสตร์



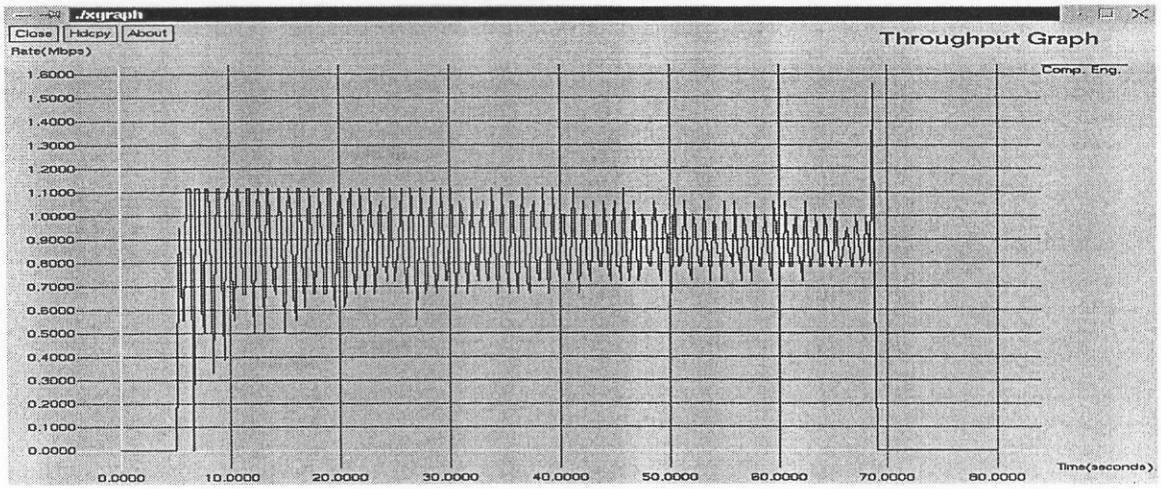
(ข) หลังการจัดคิวแบบ WFQ



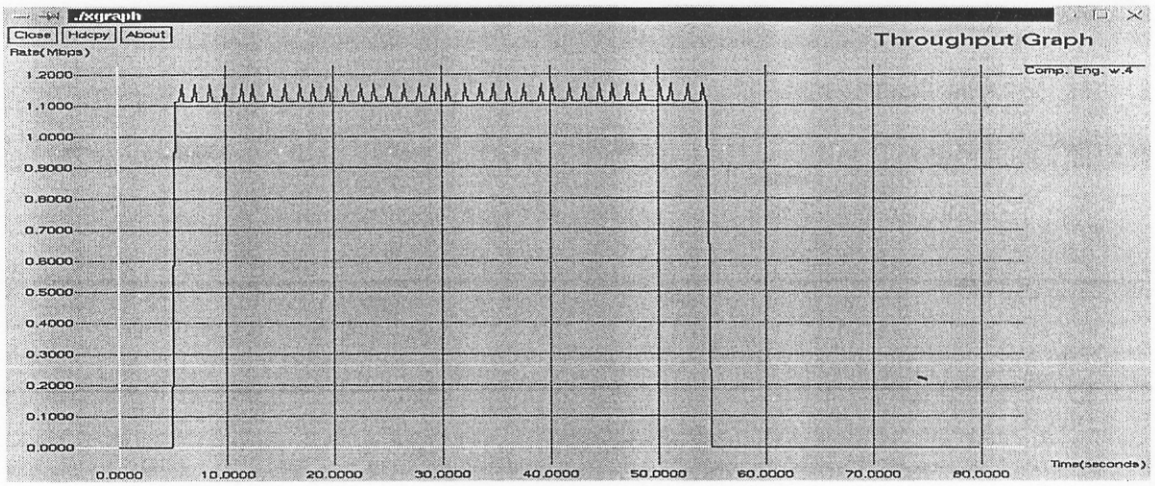
(ค) หลังการจัดคิวแบบ PQ



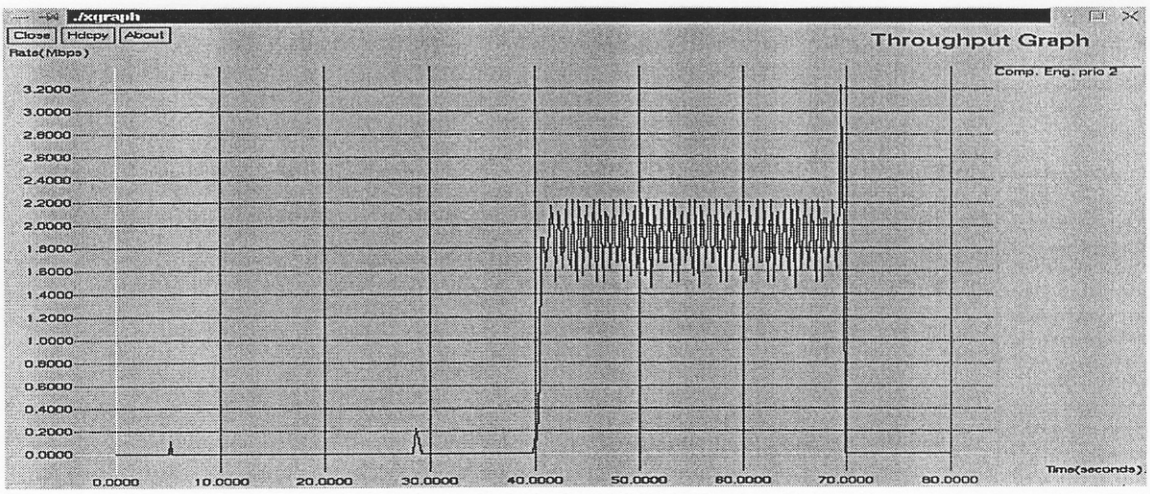
(ก) ก่อนการจัดคิวของภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์



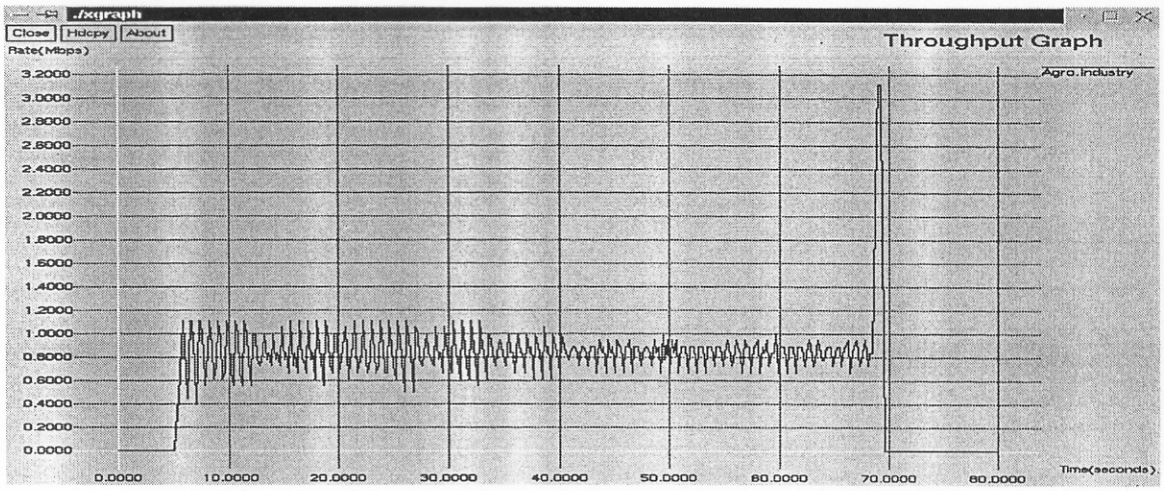
(ข) หลังการจัดคิวแบบ WFQ



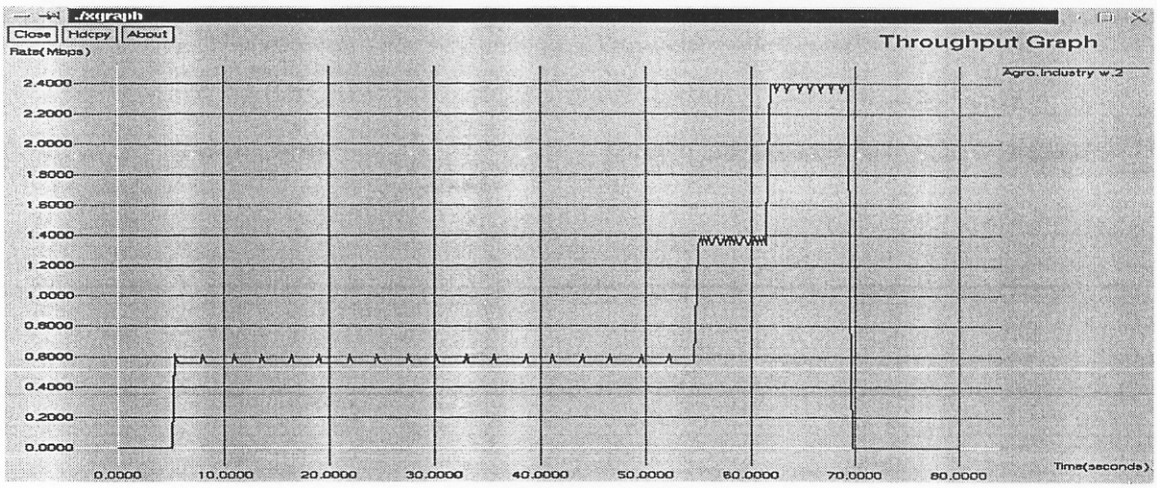
(ค) หลังการจัดคิวแบบ PQ



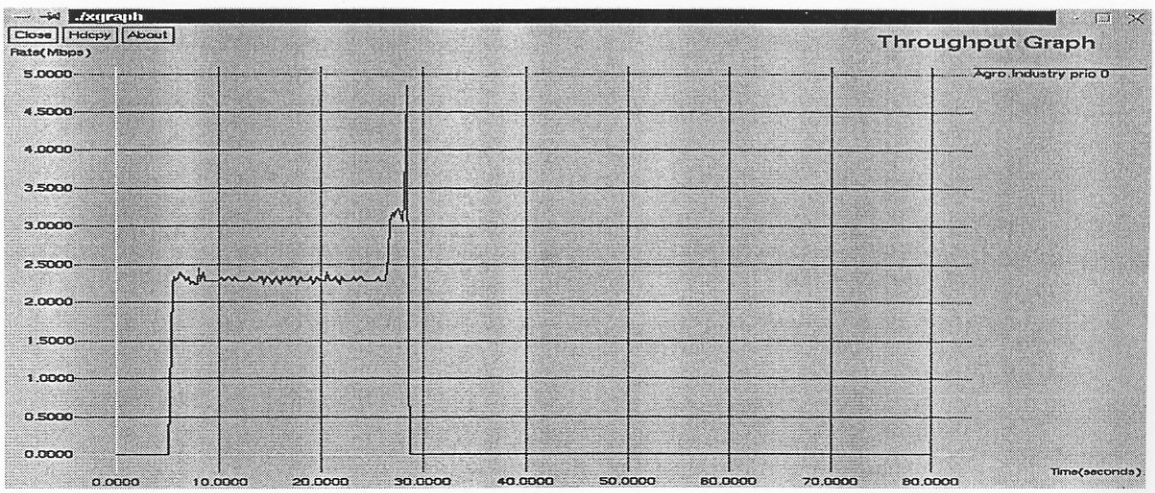
(ก) ก่อนการจัดคิวของคณะอุตสาหกรรมเกษตร



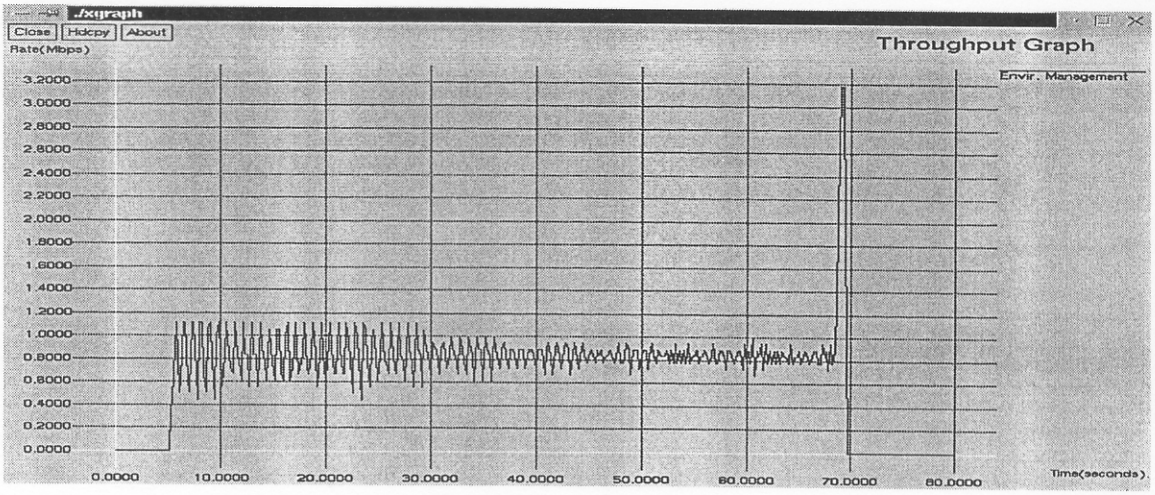
(ข) หลังการจัดคิวแบบ WFQ



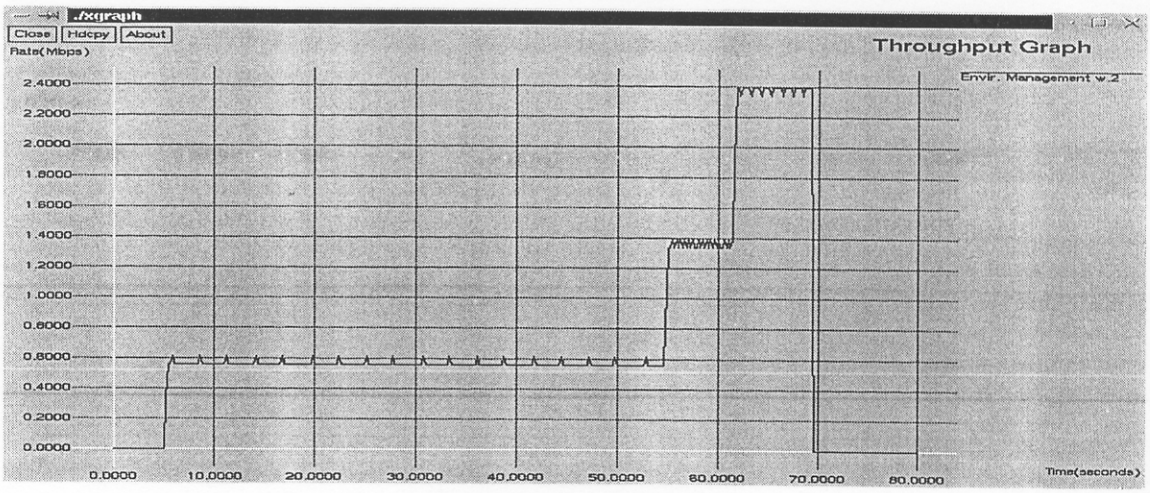
(ค) หลังการจัดคิวแบบ PQ



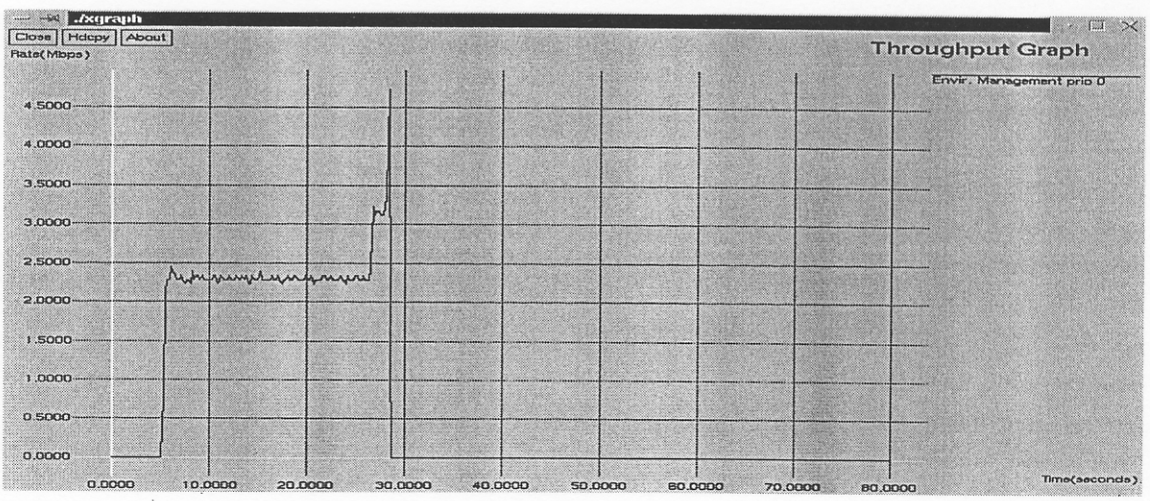
(ก) ก่อนการจัดคิวของคณะการจัดการสิ่งแวดล้อม



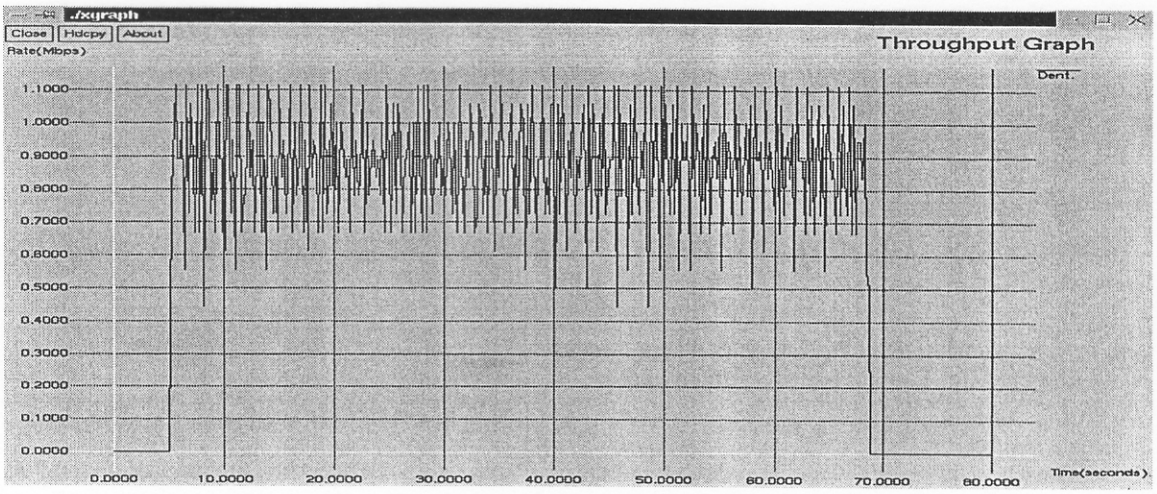
(ข) หลังการจัดคิวแบบ WFQ



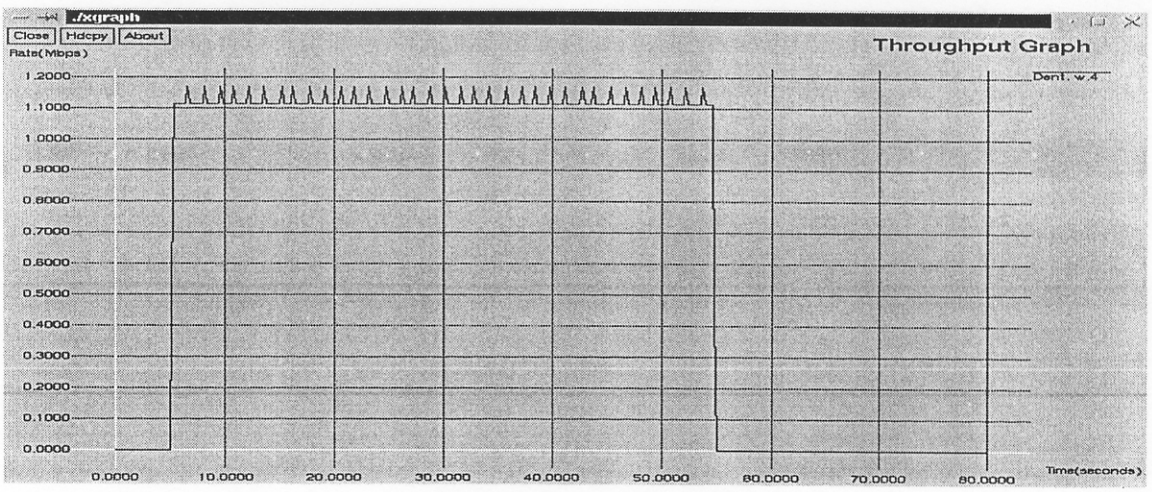
(ค) หลังการจัดคิวแบบ PQ



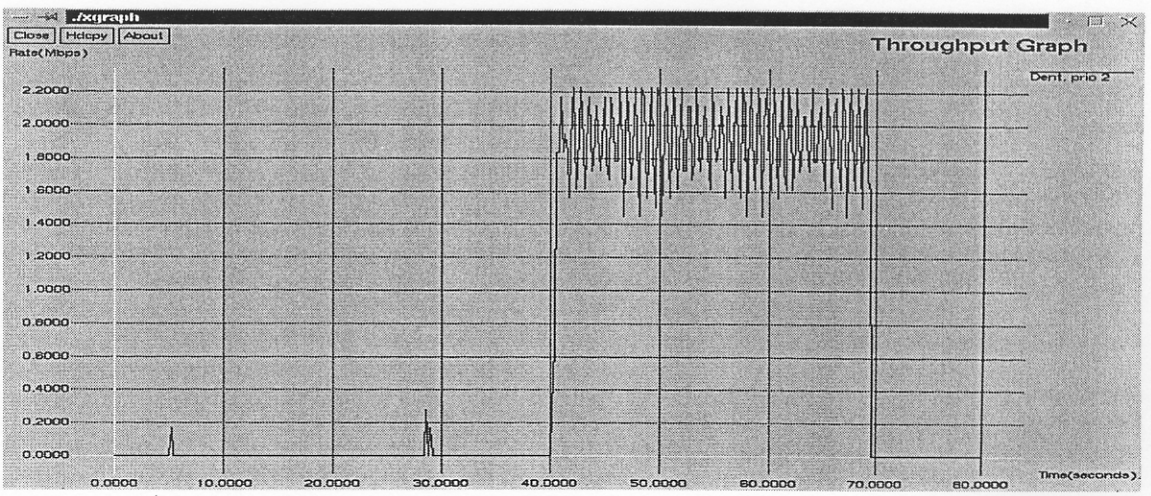
(ก) ก่อนการจัดคิวของคณะทันตแพทยศาสตร์



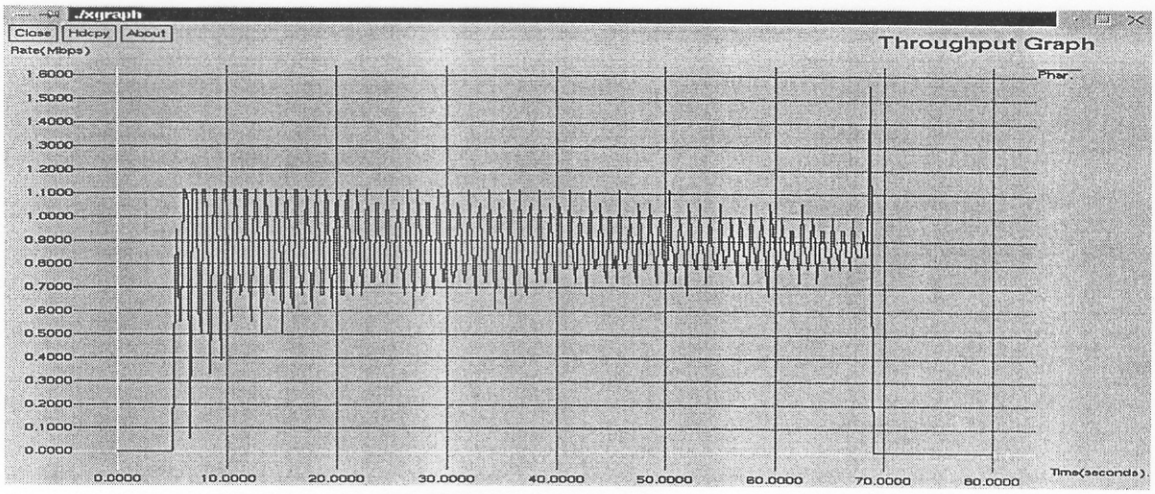
(ข) หลังการจัดคิวแบบ WFQ



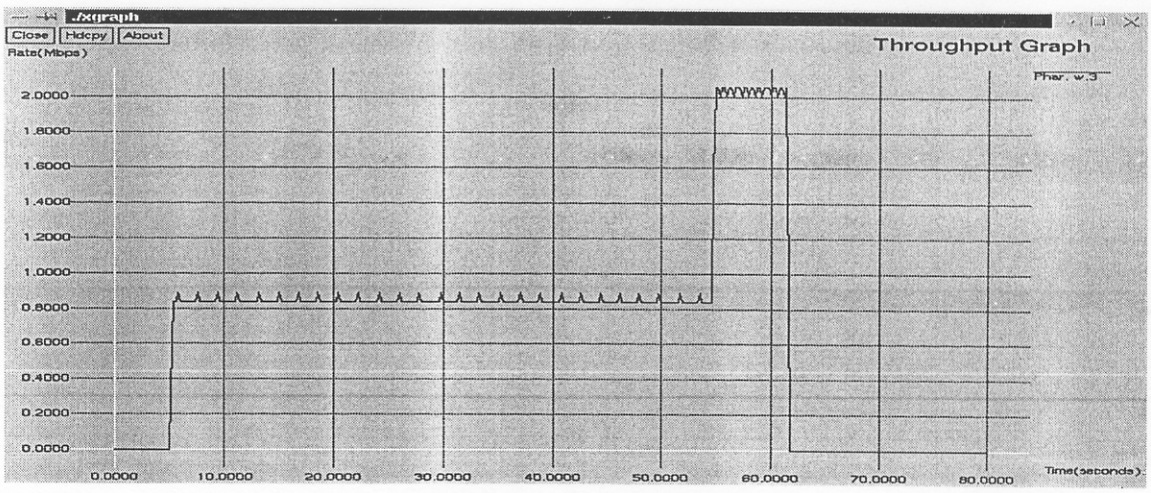
(ค) หลังการจัดคิวแบบ PQ



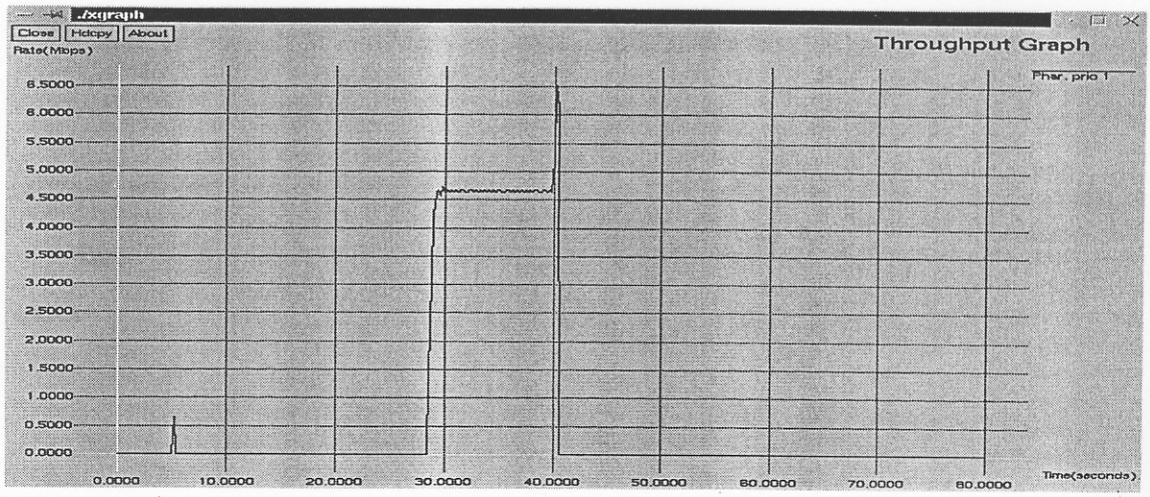
(ก) ก่อนการจัดคิวของคณะเภสัชศาสตร์



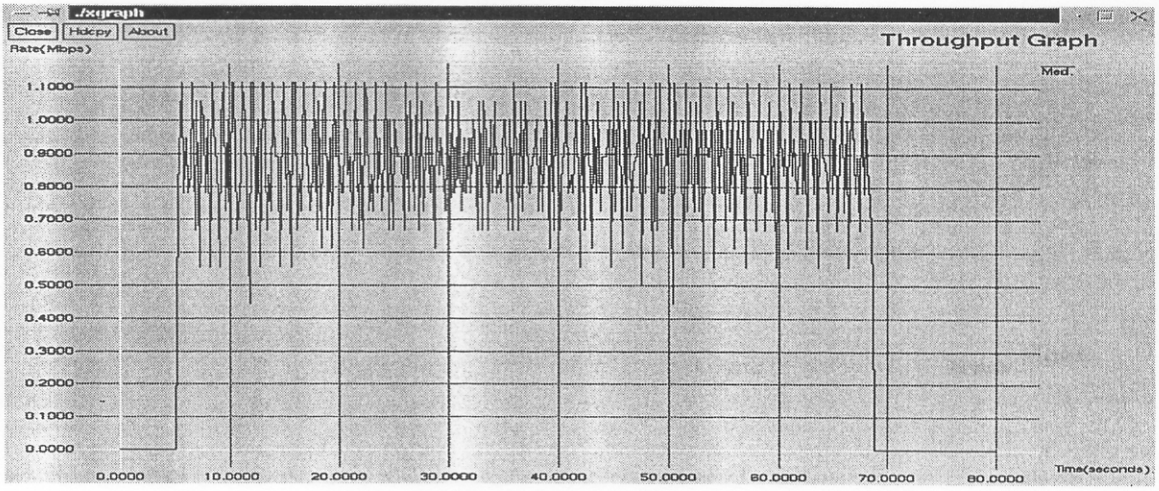
(ข) หลังการจัดคิวแบบ WFQ



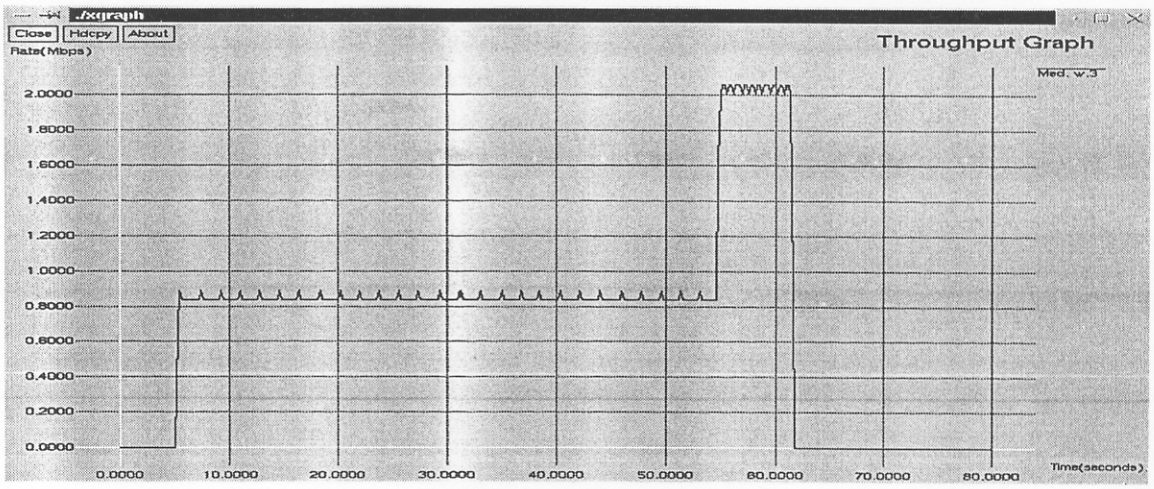
(ค) หลังการจัดคิวแบบ PQ



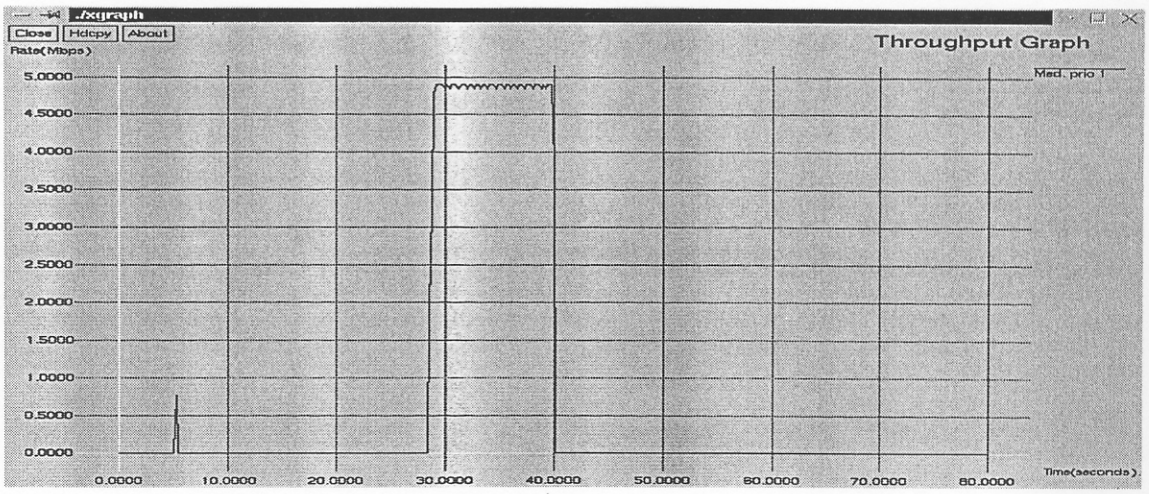
(ก) ก่อนการจัดคิวของคณะแพทยศาสตร์



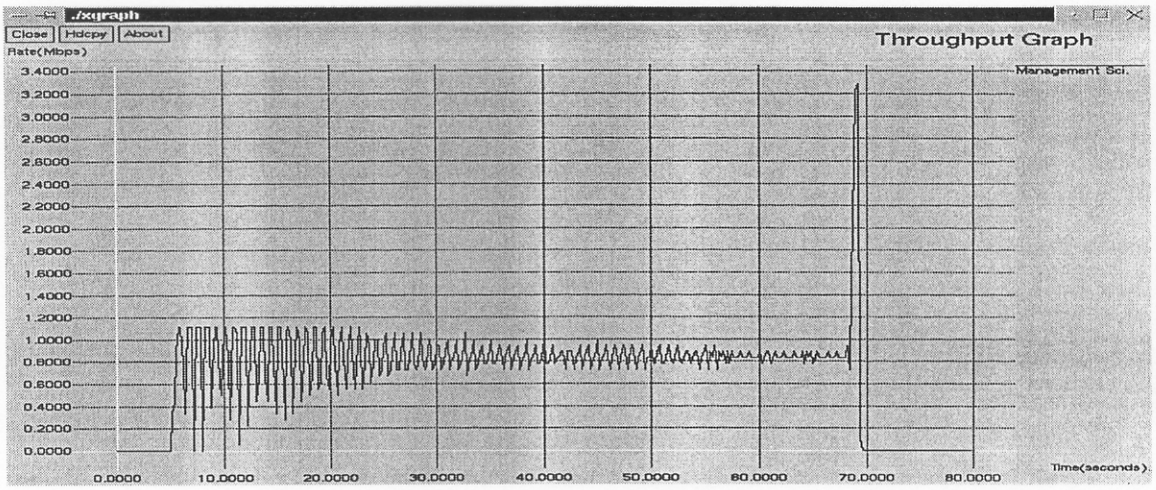
(ข) หลังการจัดคิวแบบ WFQ



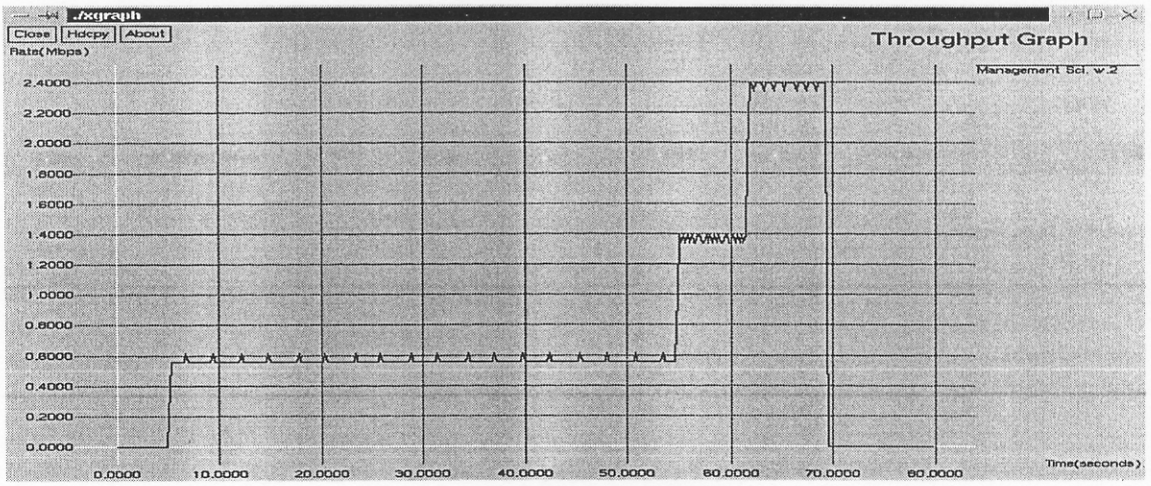
(ค) หลังการจัดคิวแบบ PQ



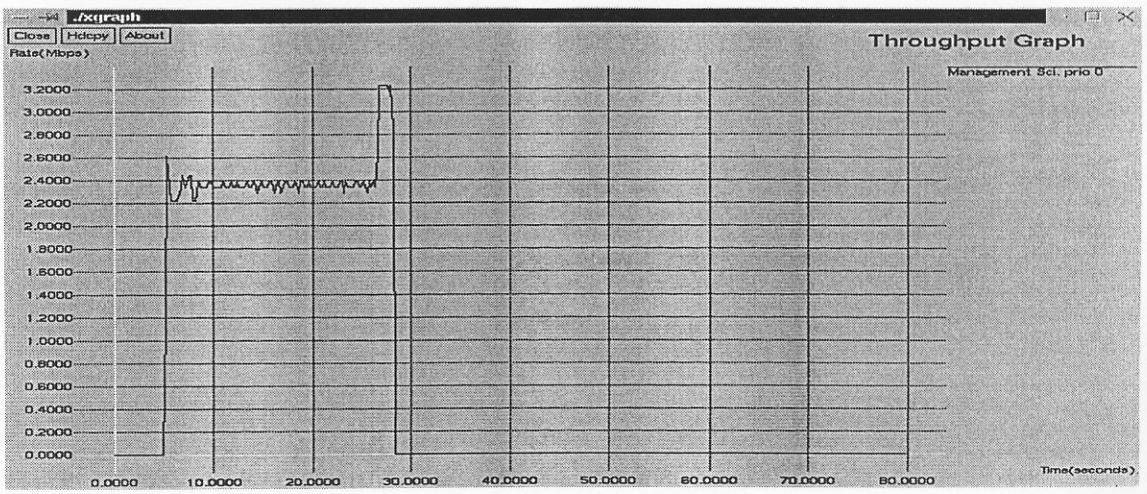
(ก) ก่อนการจัดคิวของคณะวิทยาการจัดการ



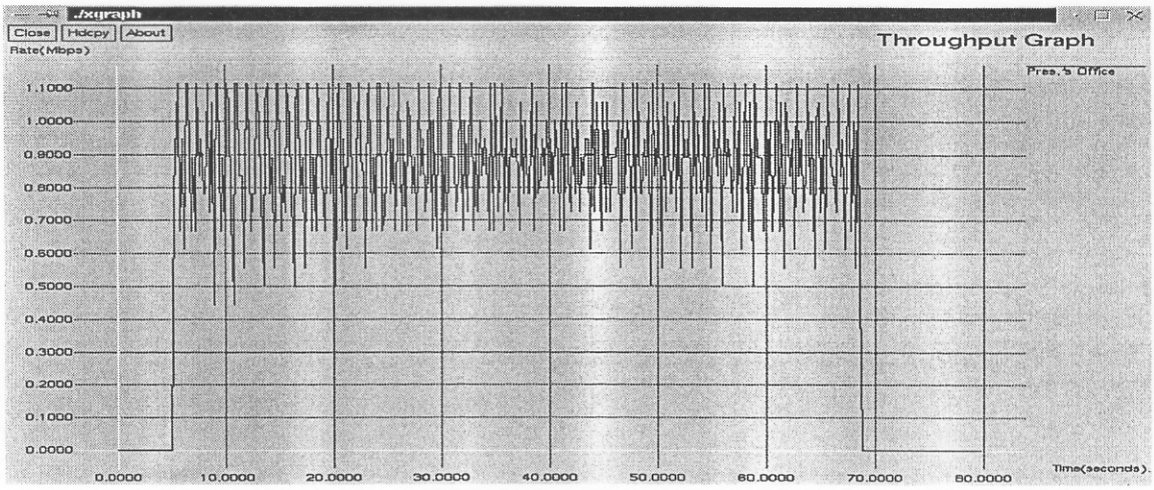
(ข) หลังการจัดคิวแบบ WFQ



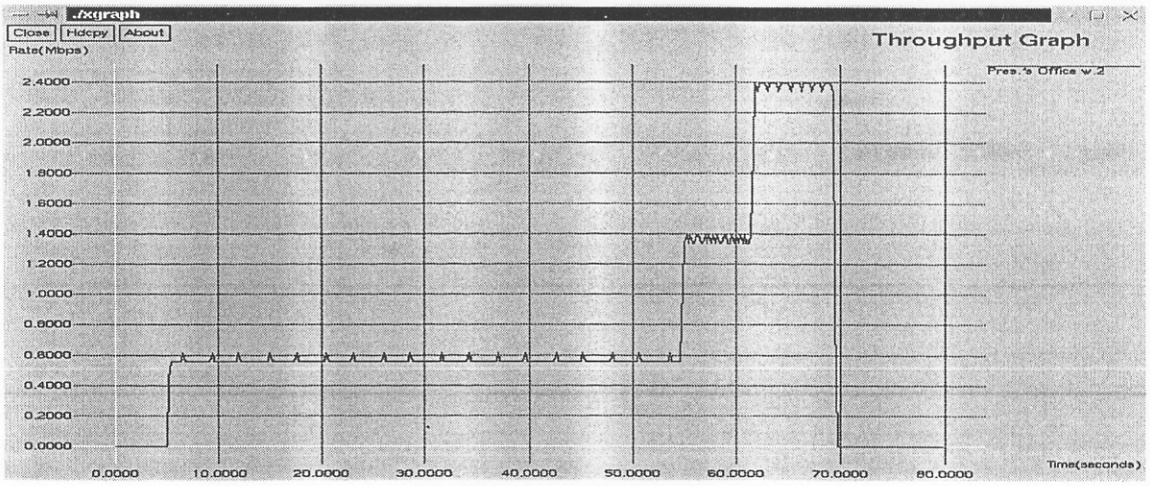
(ค) หลังการจัดคิวแบบ PQ



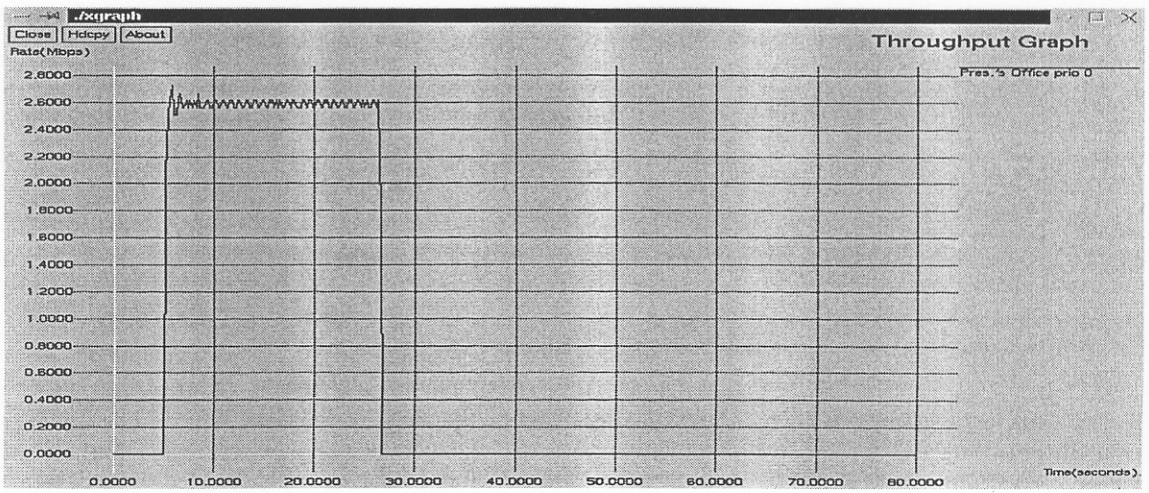
(ก) ก่อนการจัดคิวของสำนักงานอธิการบดี



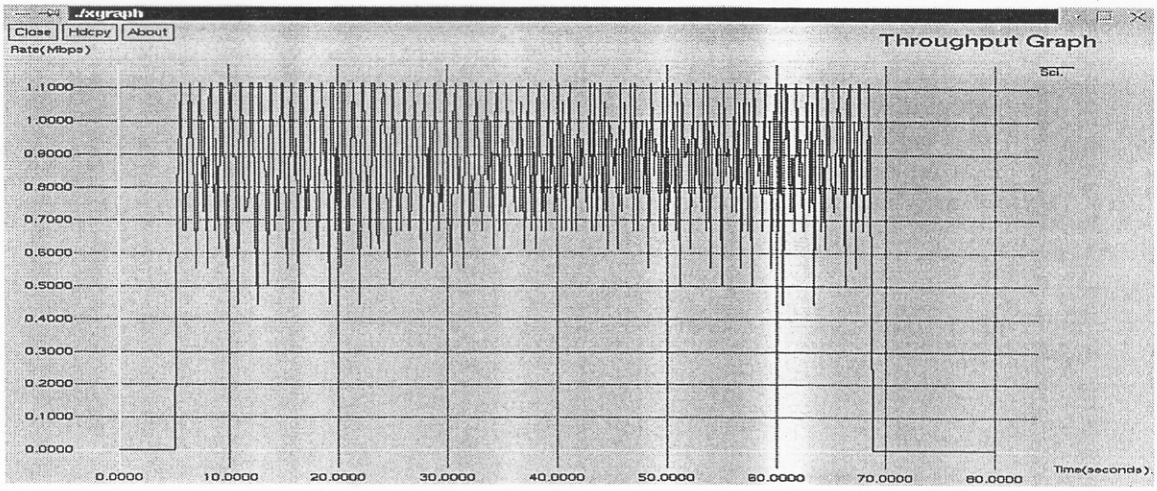
(ข) หลังการจัดคิวแบบ WFQ



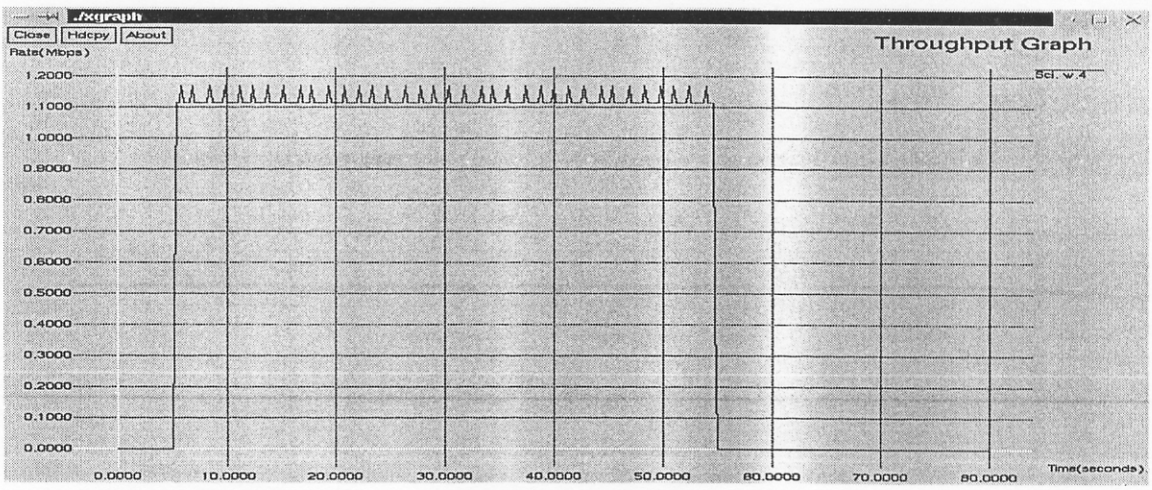
(ค) หลังการจัดคิวแบบ PQ



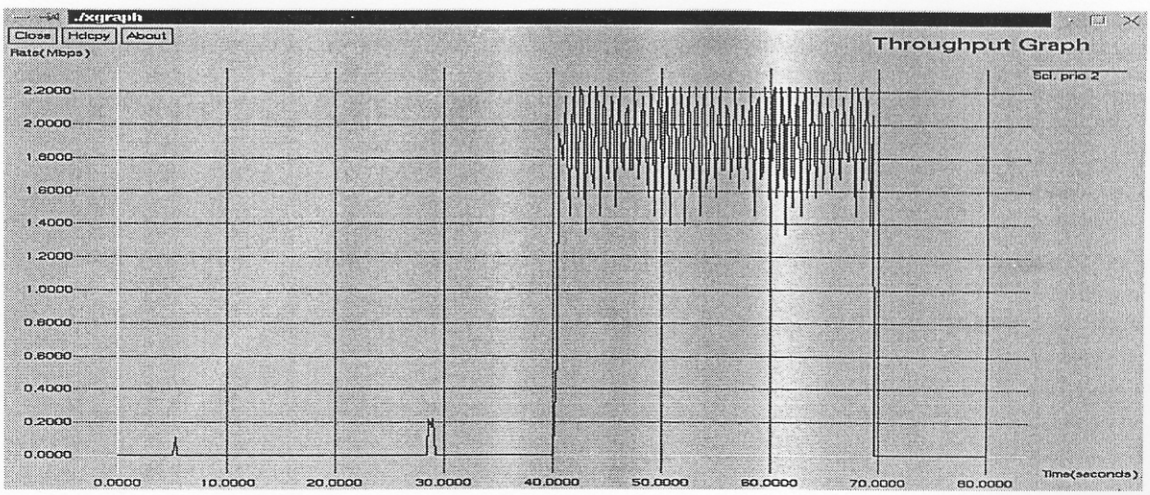
(ก) ก่อนการจัดคิวของคณะวิทยาศาสตร์



(ข) หลังการจัดคิวแบบ WFQ

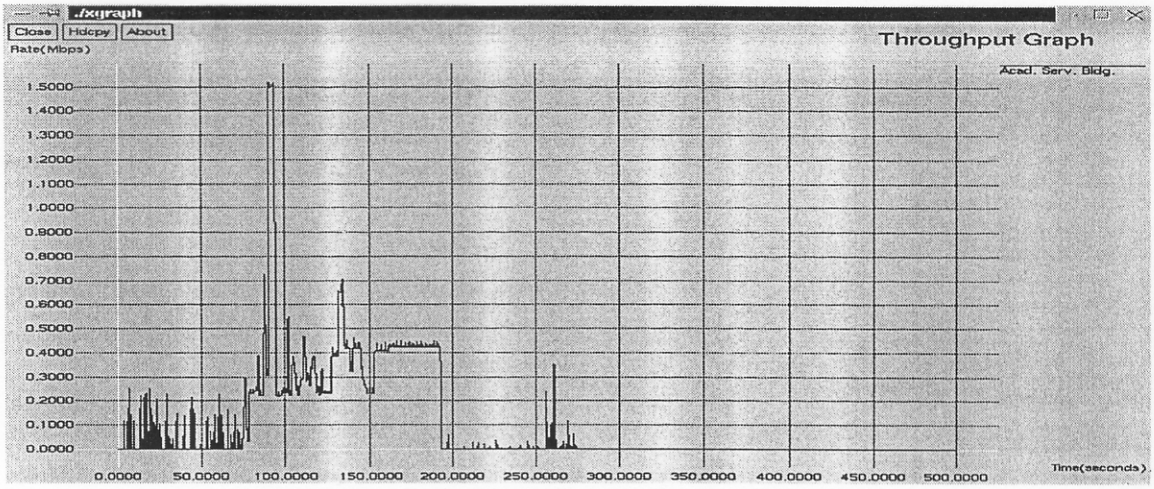


(ค) หลังการจัดคิวแบบ PQ

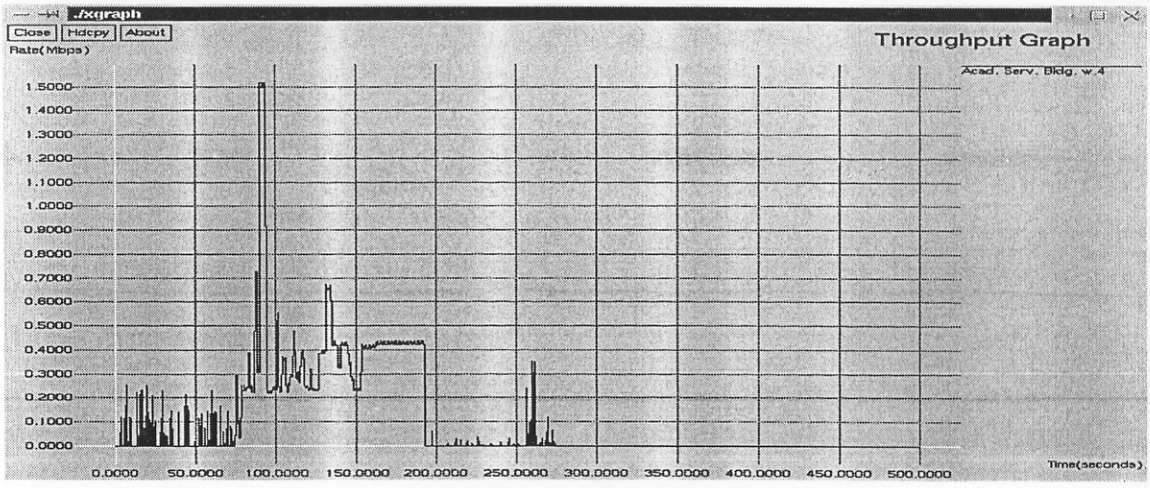


ภาคผนวก ง ผลการทดสอบของแต่ละคณะ/หน่วยงานก่อนและหลังการจัดคิว จากข้อมูลจริง

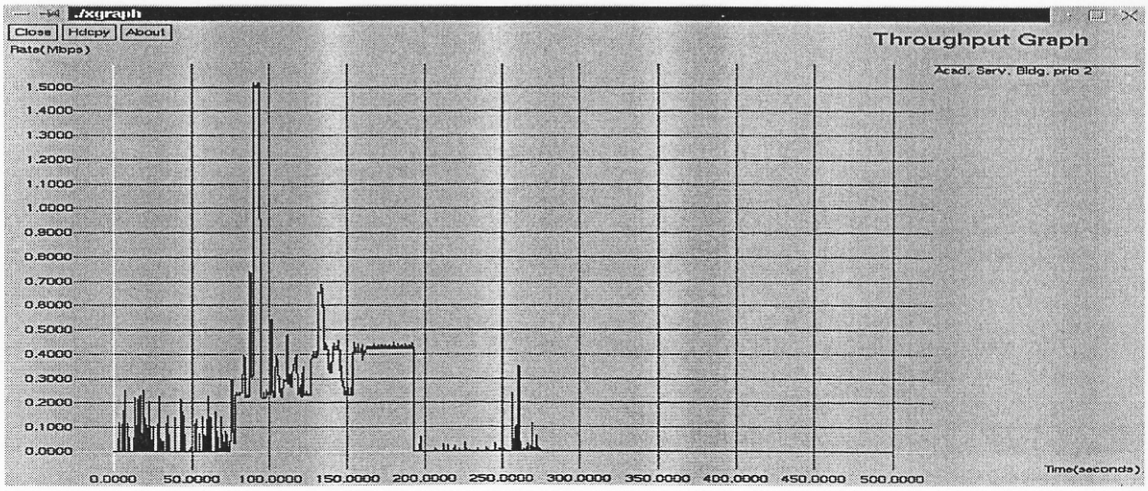
(ก) ก่อนการจัดคิวของอาคารบริหารวิชาการรวม



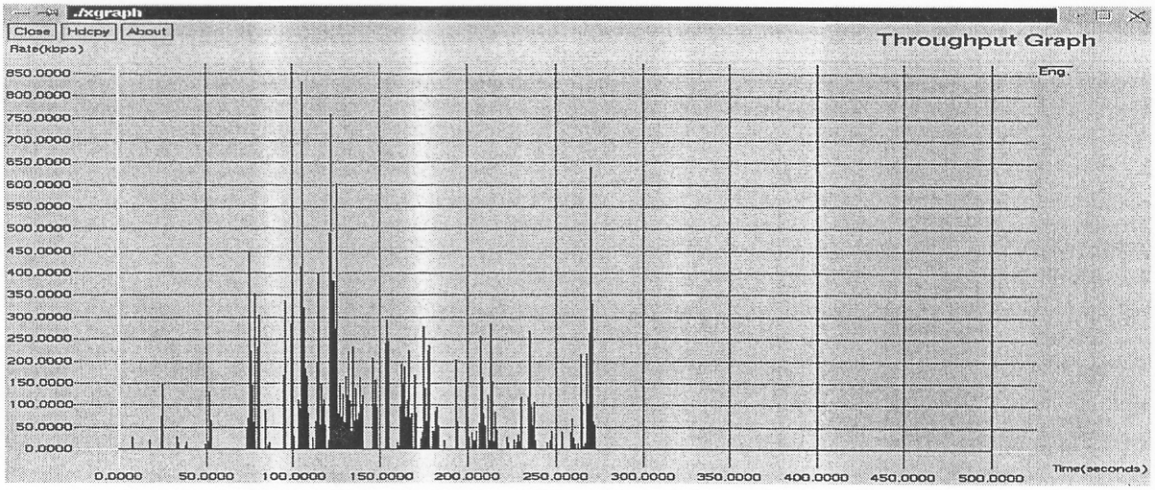
(ข) หลังการจัดคิวแบบ WFQ



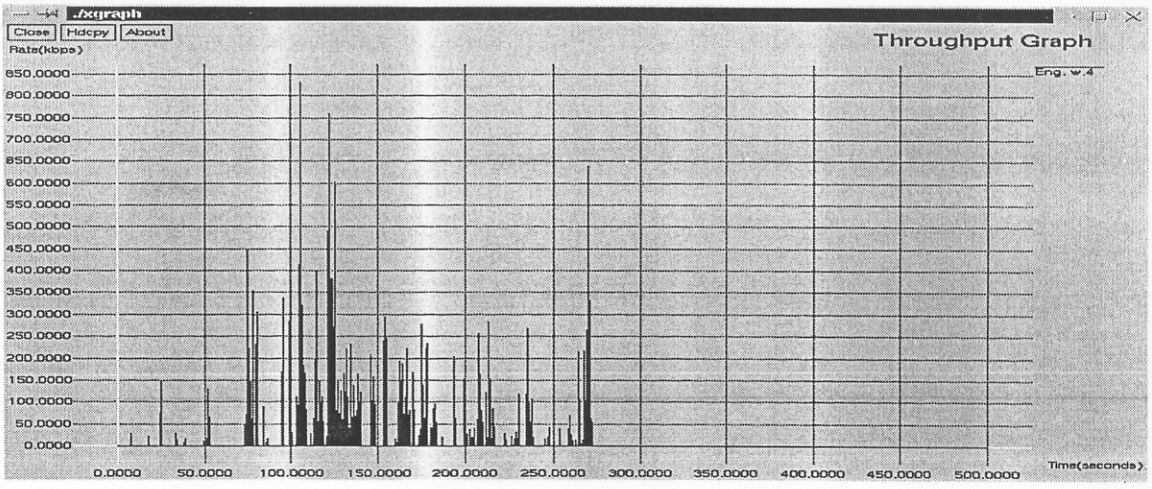
(ค) หลังการจัดคิวแบบ PQ



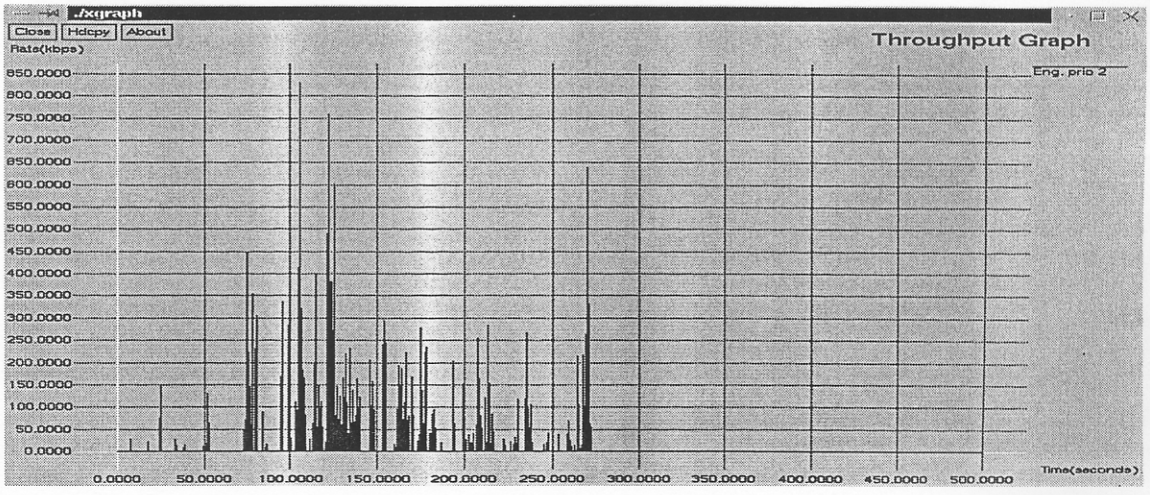
(ก) ก่อนการจัดคิวของคณะวิศวกรรมศาสตร์



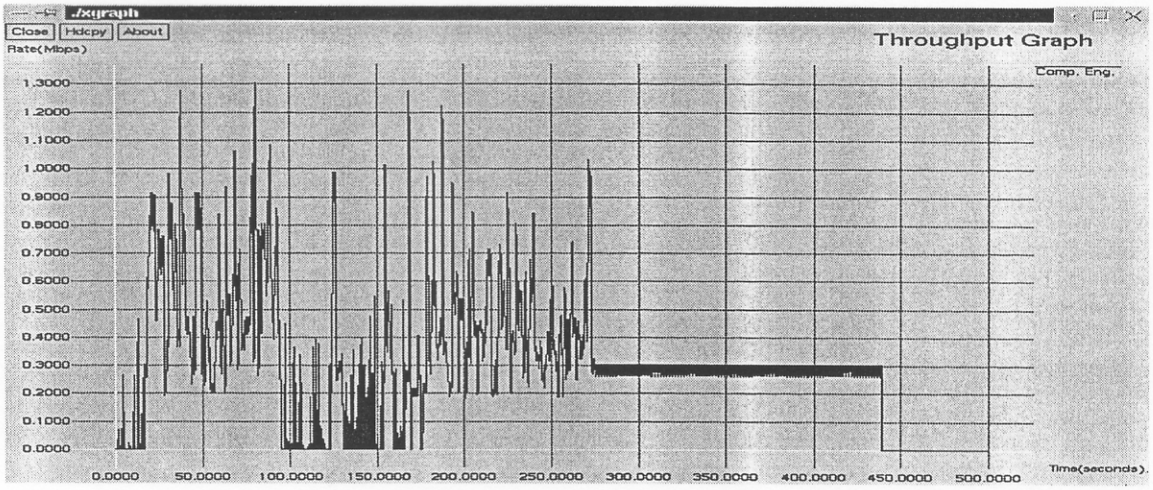
(ข) หลังการจัดคิวแบบ WFQ



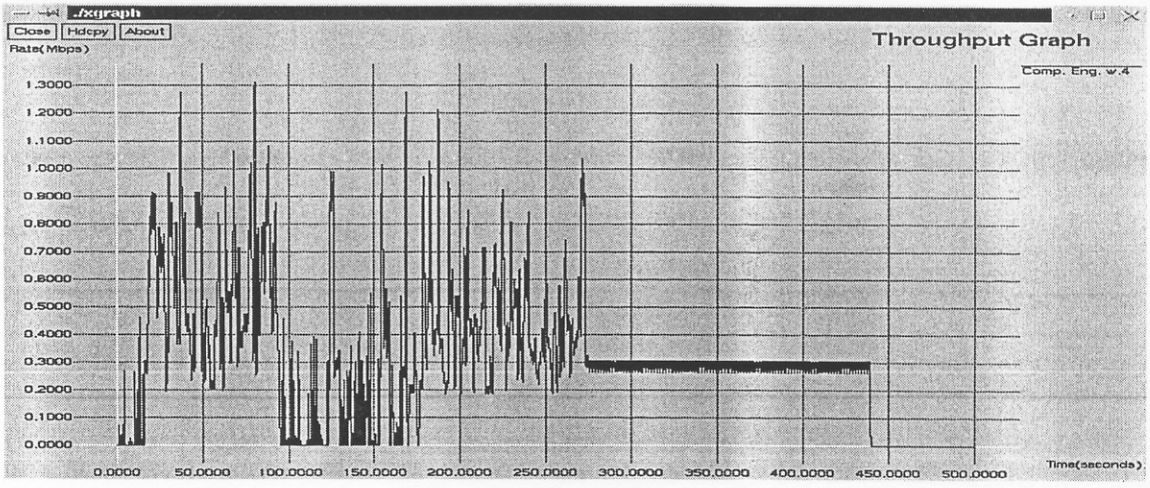
(ค) หลังการจัดคิวแบบ PQ



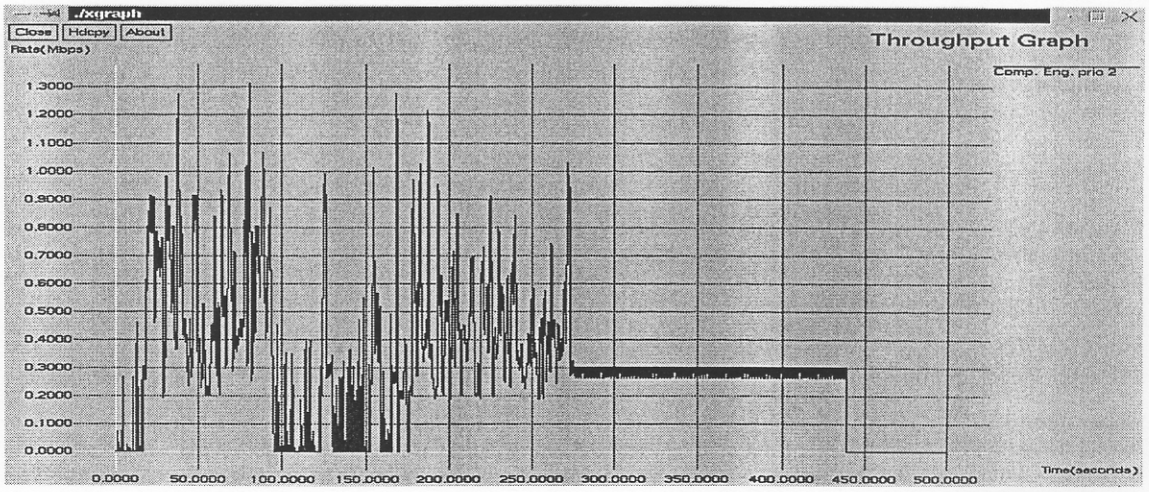
(ก) ก่อนการจัดคิวของภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์



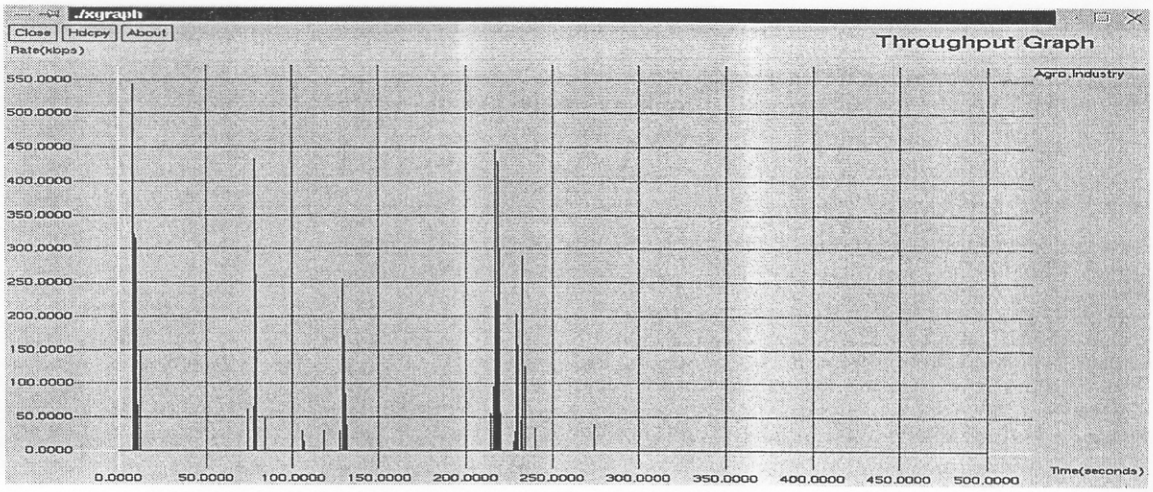
(ข) หลังการจัดคิวแบบ WFQ



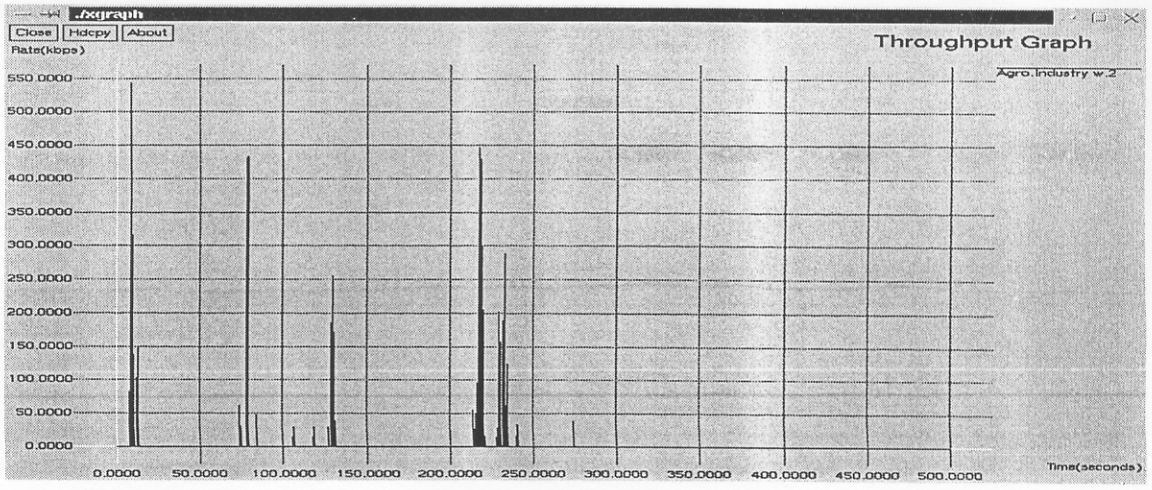
(ค) หลังการจัดคิวแบบ PQ



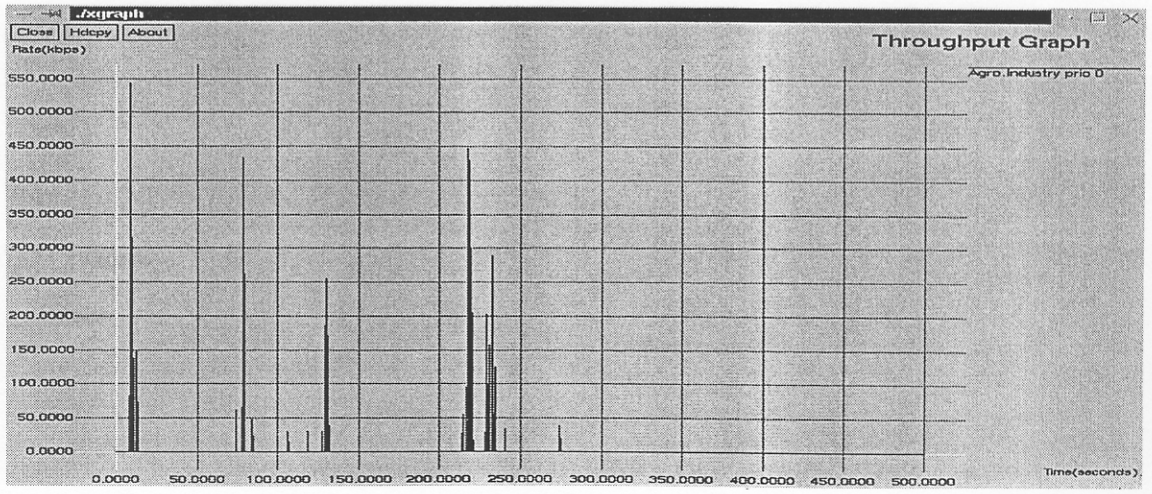
(ก) ก่อนการจัดคิวของคณะอุตสาหกรรมเกษตร



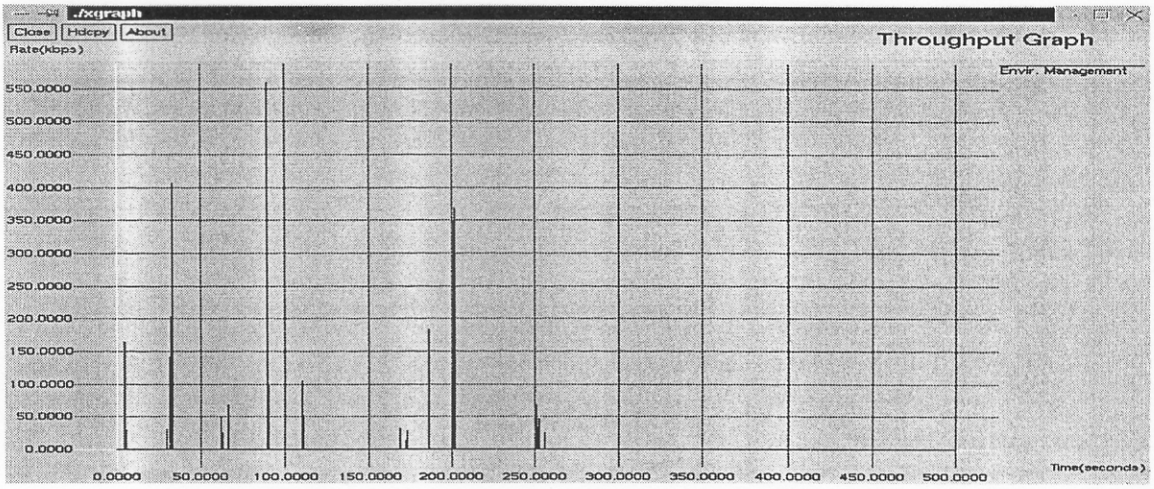
(ข) หลังการจัดคิวแบบ WFQ



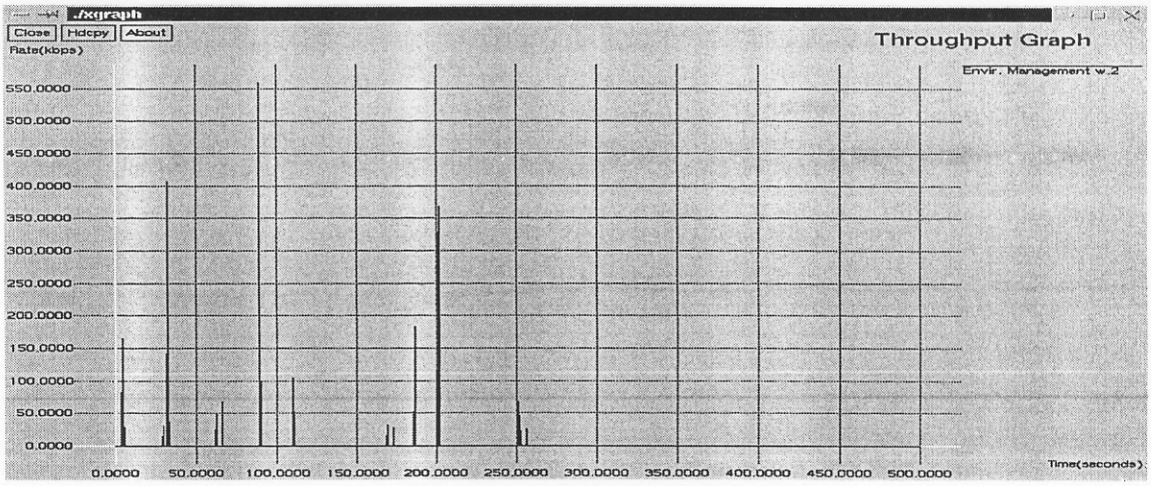
(ค) หลังการจัดคิวแบบ PQ



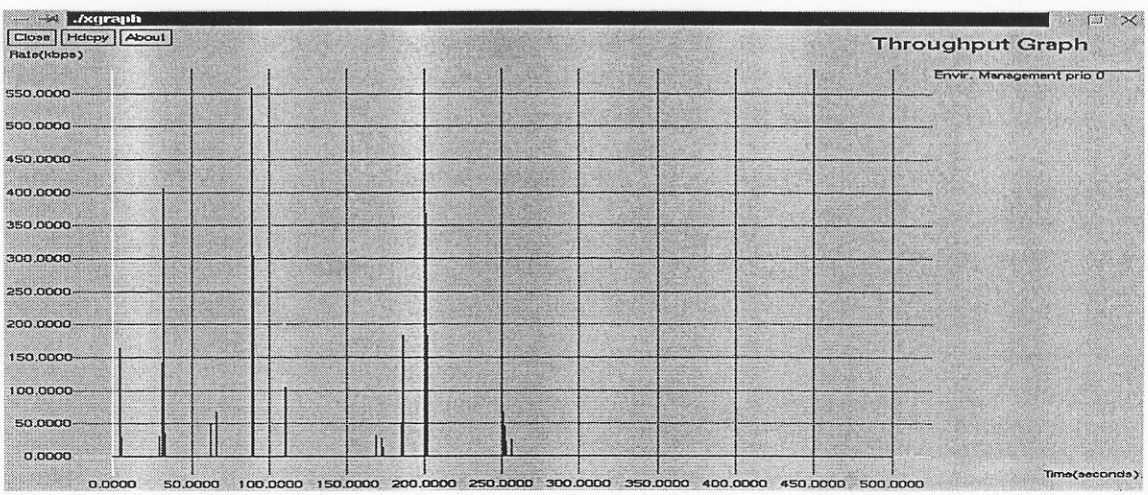
(ก) ก่อนการจัดคิวของคณะการจัดการสิ่งแวดล้อม



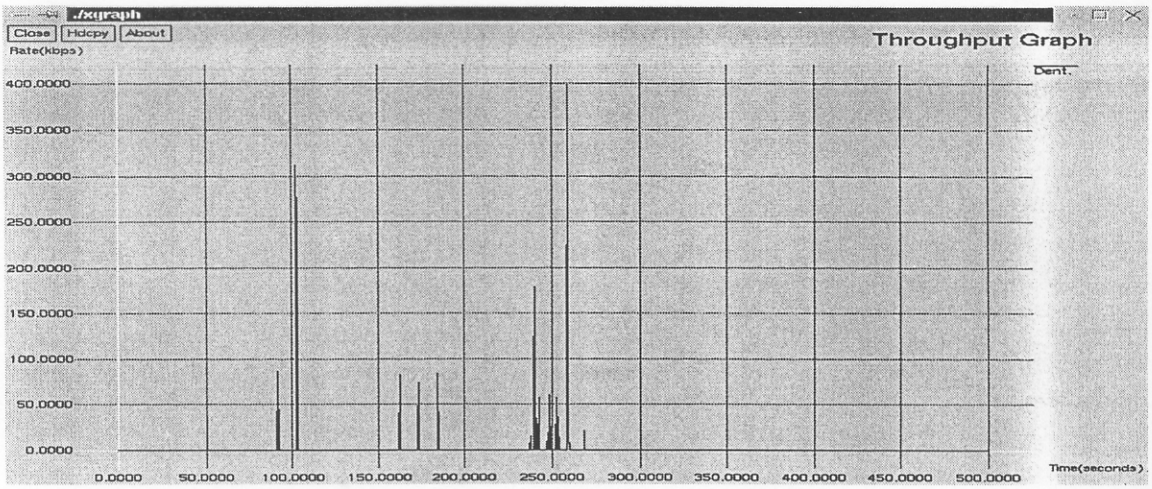
(ข) หลังการจัดคิวแบบ WFQ



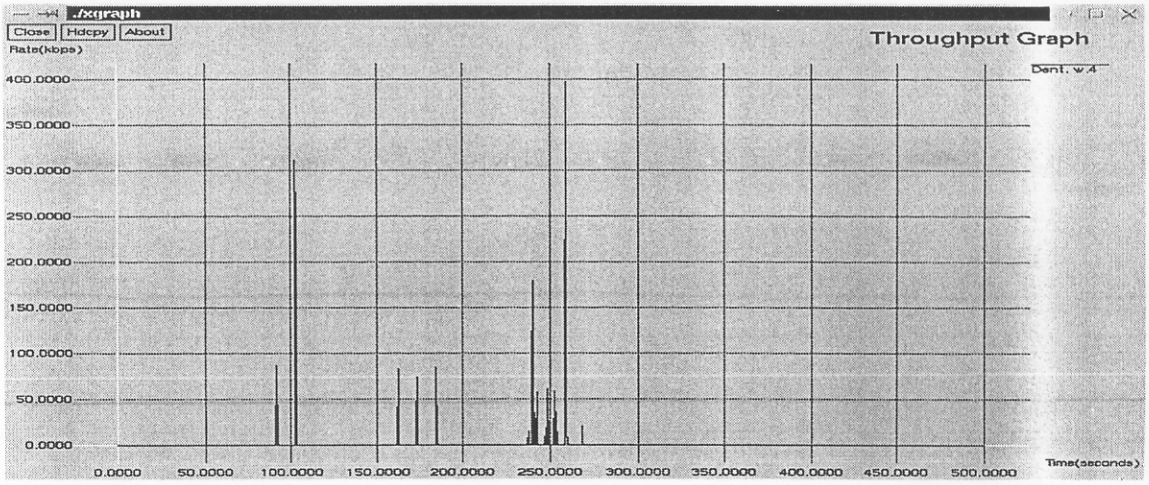
(ค) หลังการจัดคิวแบบ PQ



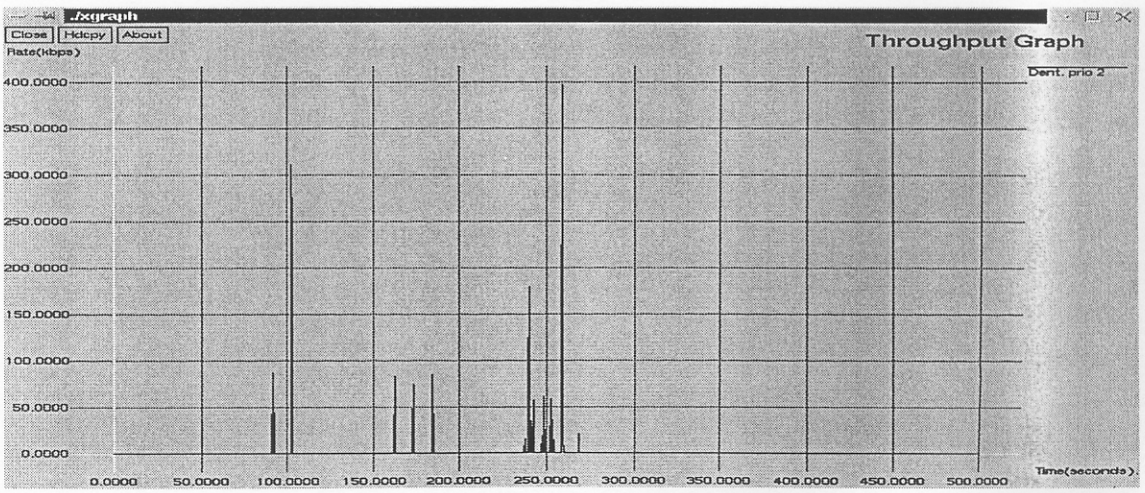
(ก) ก่อนการจัดคิวของคณะทันตแพทยศาสตร์



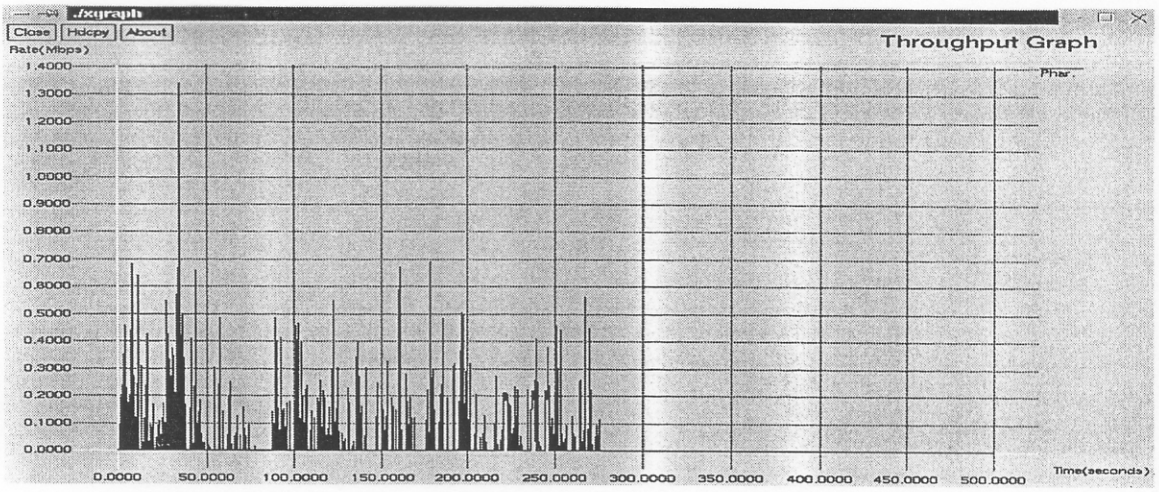
(ข) หลังการจัดคิวแบบ WFQ



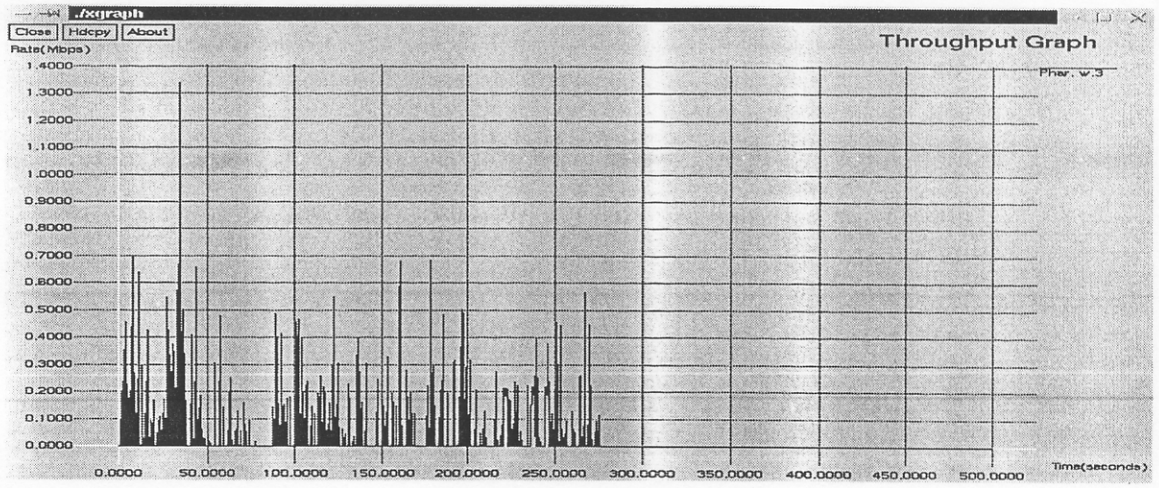
(ค) หลังการจัดคิวแบบ PQ



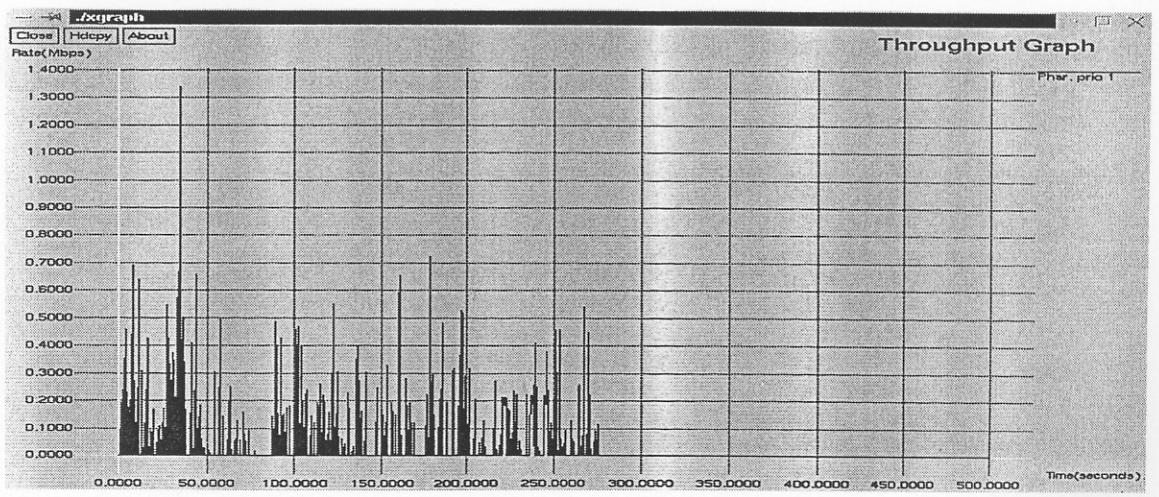
(ก) ก่อนการจัดคิวของคณะเภสัชศาสตร์



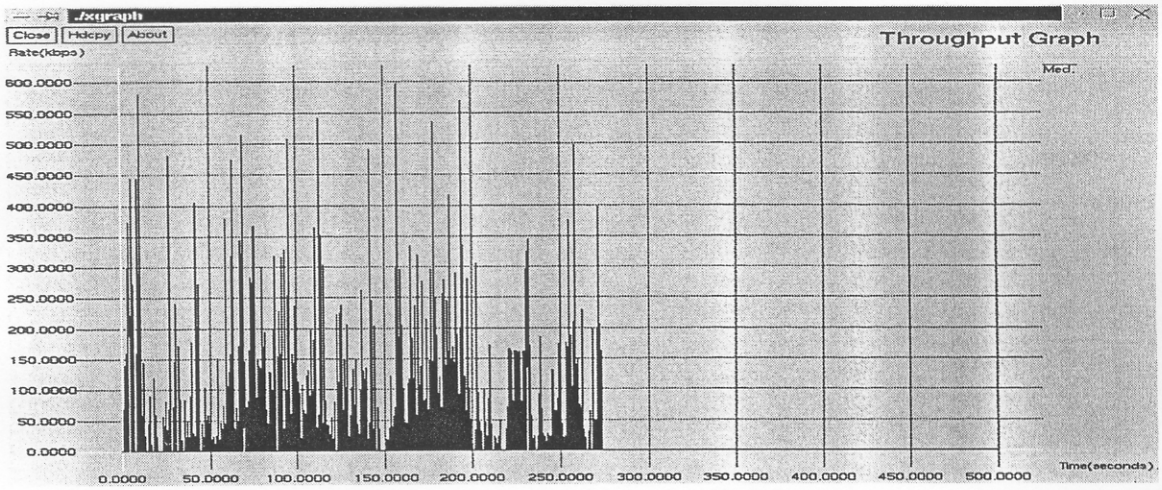
(ข) หลังการจัดคิวแบบ WFQ



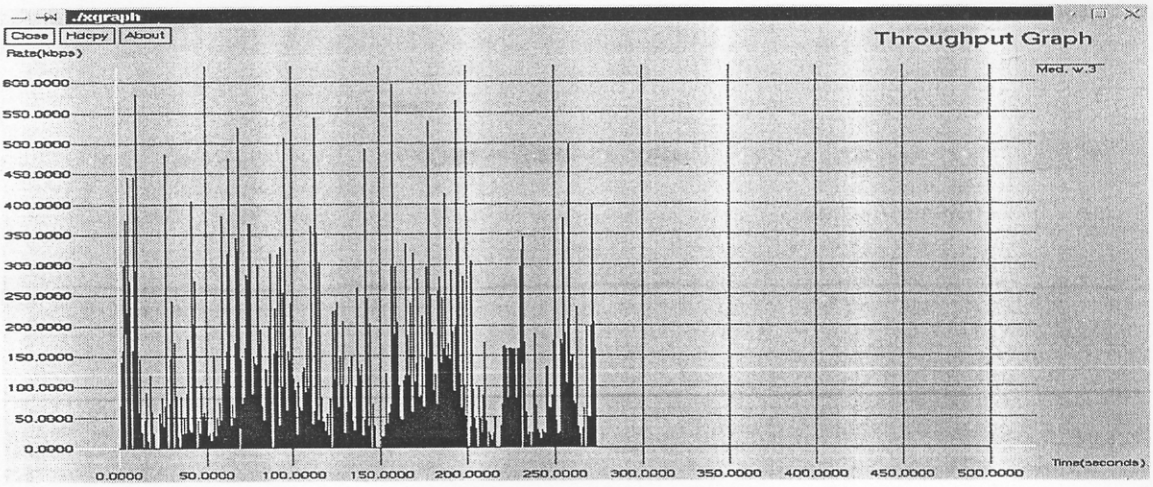
(ค) หลังการจัดคิวแบบ PQ



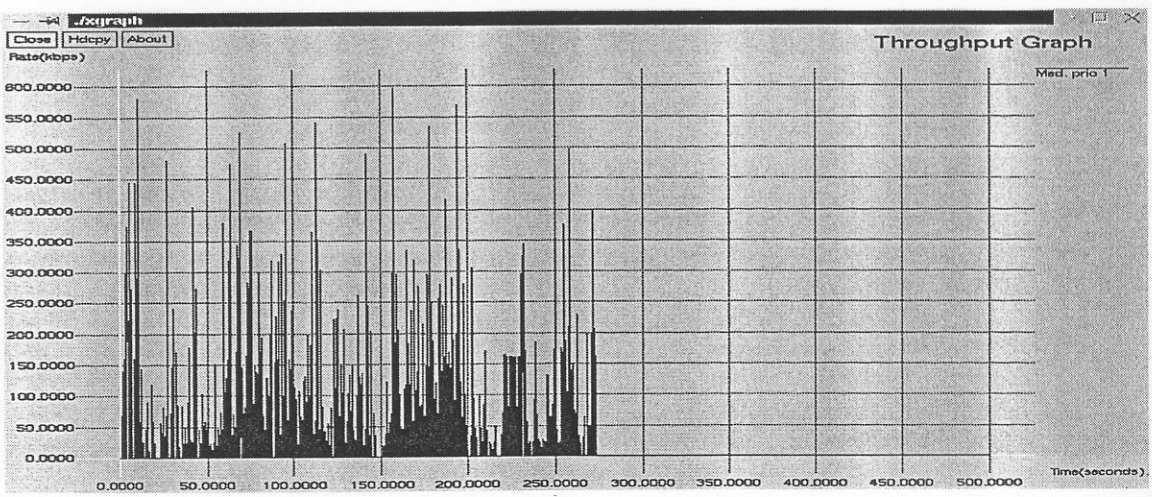
(ก) ก่อนการจัดคิวของคณะแพทยศาสตร์



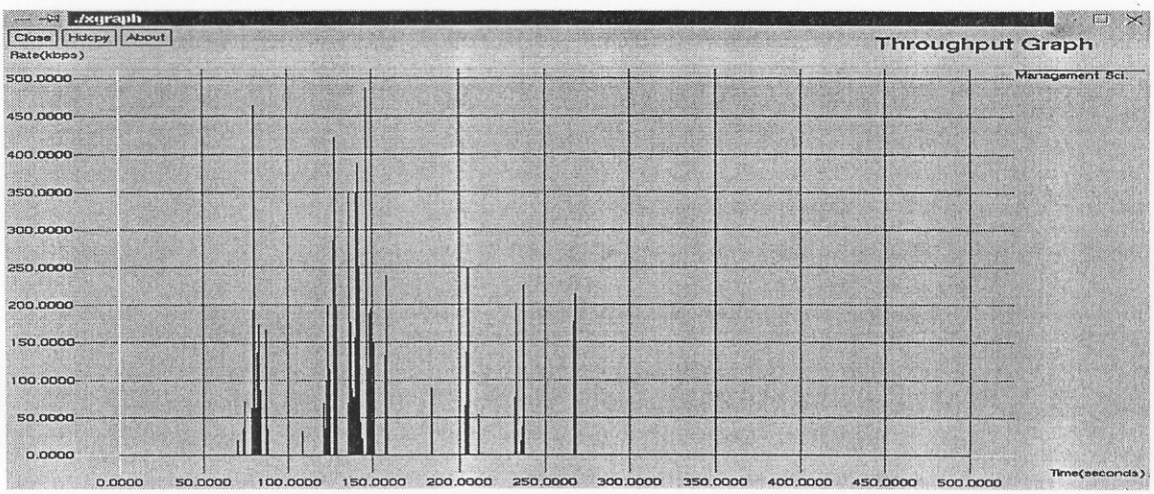
(ข) หลังการจัดคิวแบบ WFQ



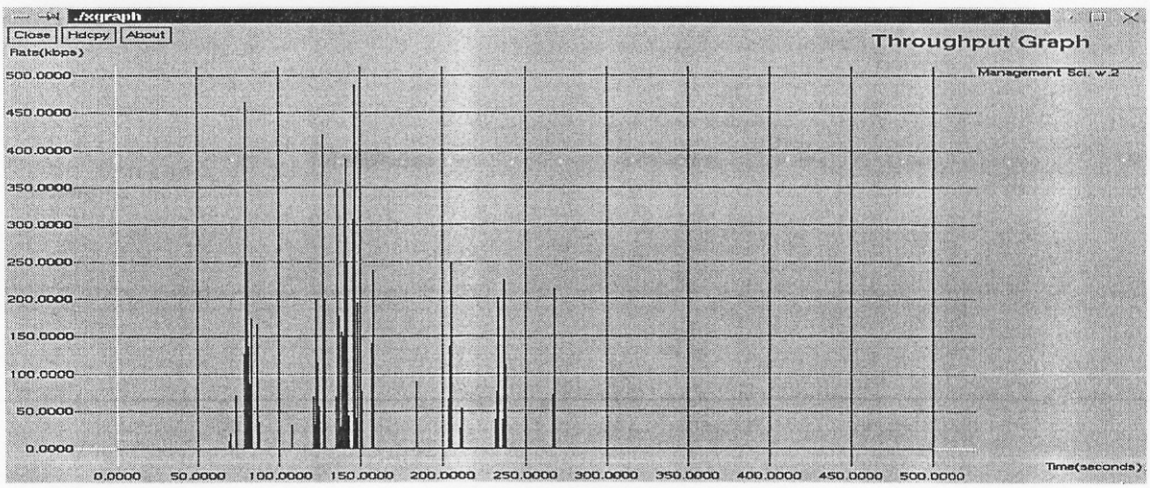
(ค) หลังการจัดคิวแบบ PQ



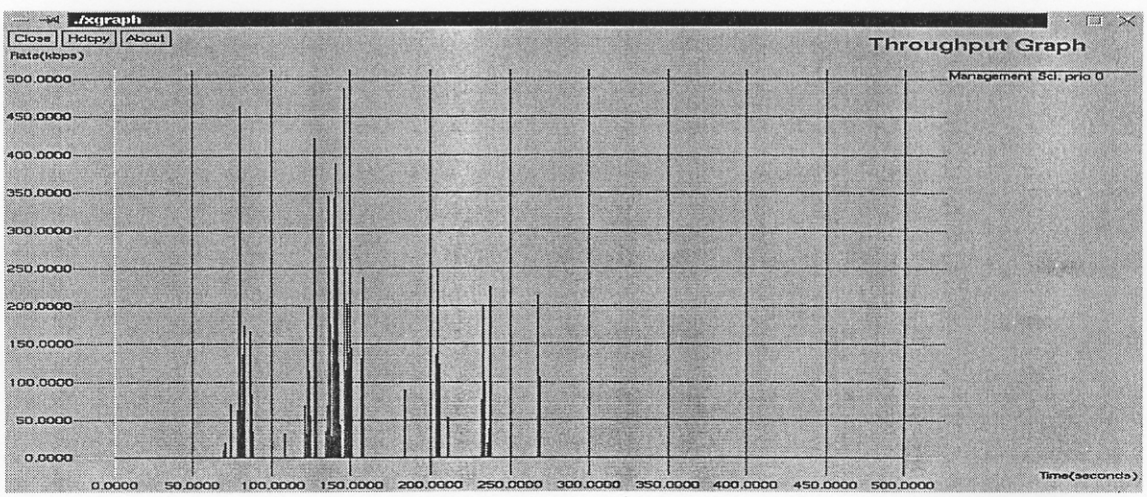
(ก) ก่อนการจัดคิวของคณะวิทยาการจัดการ



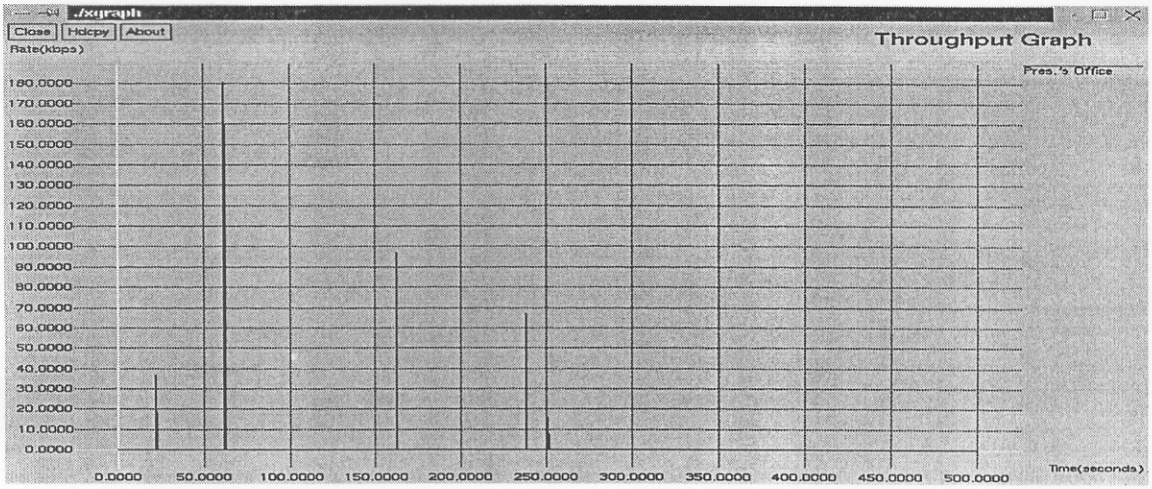
(ข) หลังการจัดคิวแบบ WFQ



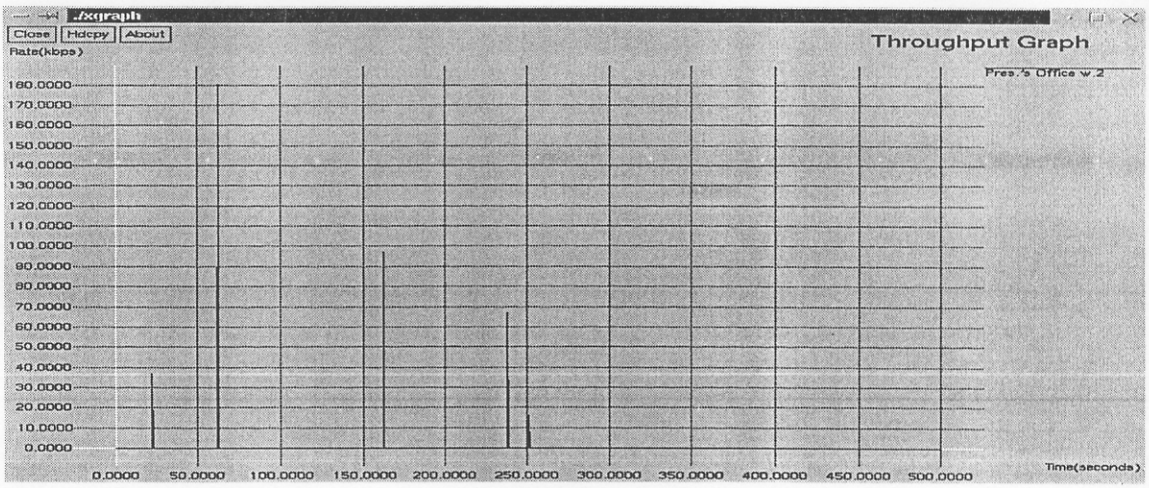
(ค) หลังการจัดคิวแบบ PQ



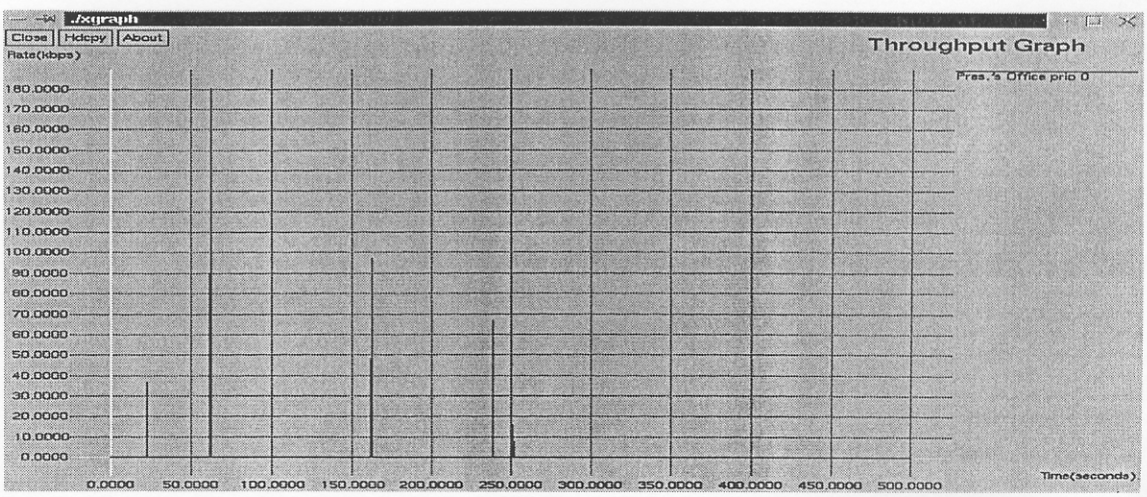
(ก) ก่อนการจัดคิวของสำนักงานอธิการบดี



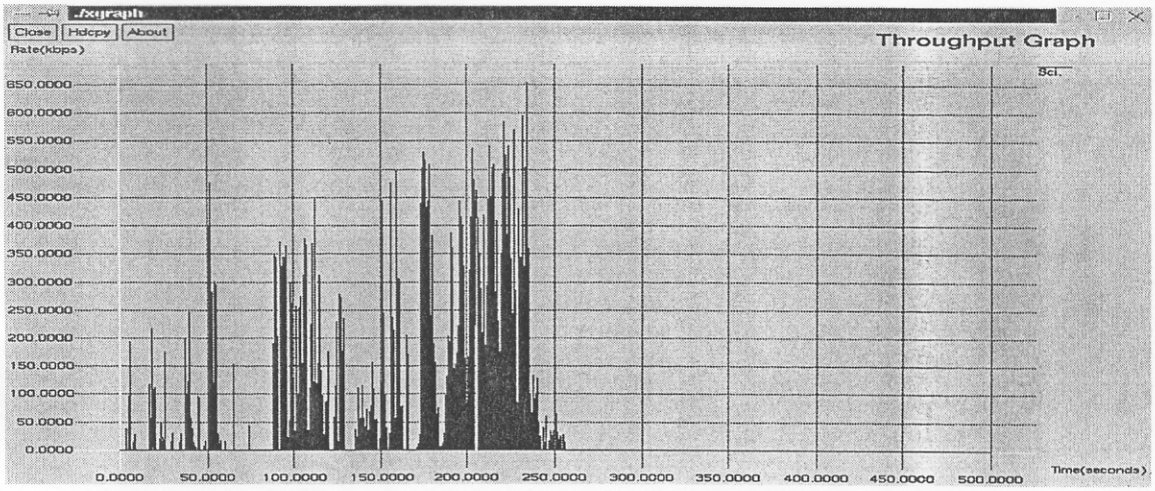
(ข) หลังการจัดคิวแบบ WFQ



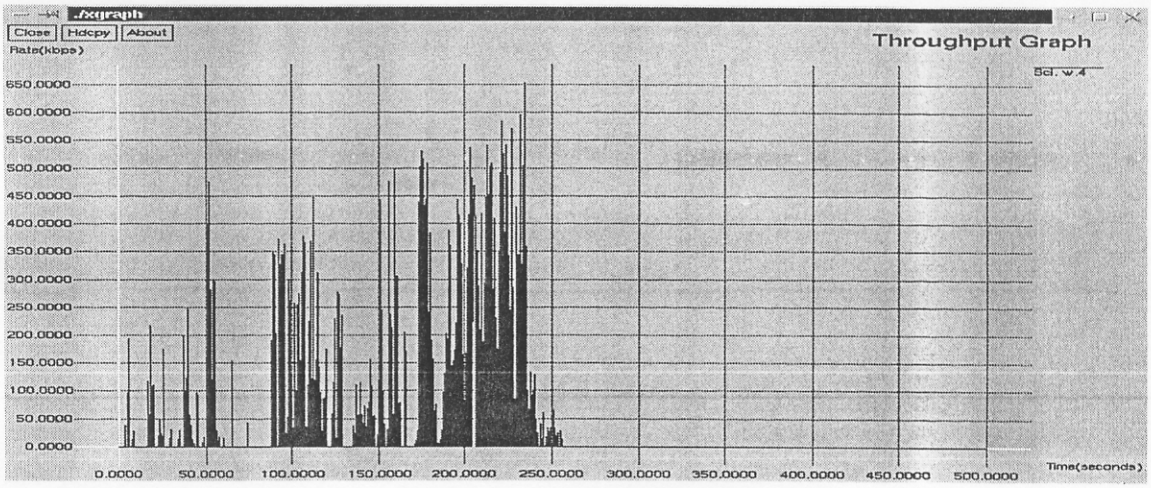
(ค) หลังการจัดคิวแบบ PQ



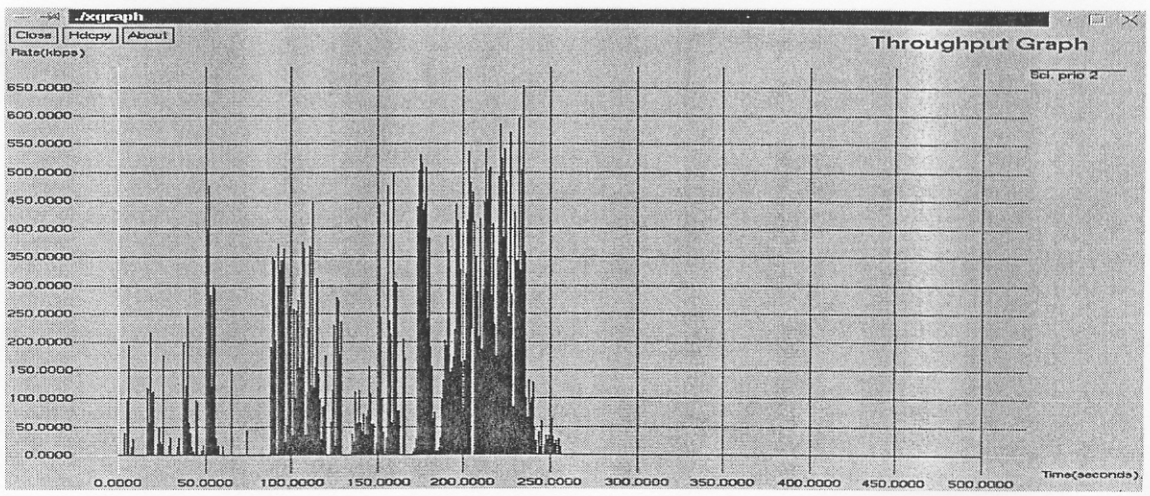
(ก) ก่อนการจัดคิวของคณะวิทยาศาสตร์



(ข) หลังการจัดคิวแบบ WFQ



(ค) หลังการจัดคิวแบบ PQ



ภาคผนวก จ ผลงานทางวิชาการที่ได้รับการตีพิมพ์

ที่ ทม 5145/ว.22

มหาวิทยาลัยเทคโนโลยีสุรนารี
111 ถนนมหาวิทยาลัย ตำบลสุรนารี
อำเภอเมือง จังหวัดนครราชสีมา 30000

2 กรกฎาคม 2545

เรื่อง การนำเสนอผลงานวิจัย ในการประชุมเสนอผลงานวิจัยระดับบัณฑิตศึกษาของประเทศไทย ครั้งที่ 3

เรียน นางสาวหฤทัย สมบูรณ์รุ่งโรจน์

สิ่งที่ส่งมาด้วย 1. รายละเอียดการนำเสนอผลงานวิจัยแบบโปสเตอร์และแบบบรรยาย
2. แบบแสดงความจำนงเข้าร่วมการประชุมและการลงทะเบียน
3. กำหนดการประชุม
4. แผนผังมหาวิทยาลัยเทคโนโลยีสุรนารี

ตามที่ท่านได้รับการพิจารณาให้นำเสนอผลงานวิจัยในการประชุมเสนอผลงานวิจัยระดับบัณฑิตศึกษาของประเทศไทย ครั้งที่ 3 ระหว่างวันที่ 18 – 19 กรกฎาคม 2545 ณ มหาวิทยาลัยเทคโนโลยีสุรนารี ความละเอียดทราบแล้วนั้น

คณะอนุกรรมการฝ่ายวิชาการการจัดการประชุมฯ ขอแจ้งวันเวลาและสถานที่ในการนำเสนอ ผลงานวิจัย ดังนี้

ผลงานวิจัยของ *นางสาวหฤทัย สมบูรณ์รุ่งโรจน์* เรื่อง *การศึกษาผลกระทบของขนาดคิวของตัวจัดการแบบดิวิดท์ต่อปริมาณความแออัดของข้อมูลที่เกิดขึ้นบนระบบเครือข่าย* จัดอยู่ในกลุ่มสาขาวิจัยด้าน*วิศวกรรมศาสตร์* ได้รับการพิจารณาให้นำเสนอ*บรรยาย ในวันพฤหัสบดีที่ 18 ก.ค. 45 เวลา 10.25-12.00 น. ณ ห้อง 211 อาคารสุรพัฒน์ 1* มหาวิทยาลัยเทคโนโลยีสุรนารี

ทั้งนี้ ผลงานวิจัยของท่าน*ได้รับการตีพิมพ์*ในสื่อรวบรวมบทความทางวิชาการ เนื่องจากท่าน*ได้จัดส่ง*บทความ (Extended Abstract) มาให้ทางคณะกรรมการ

จึงเรียนมาเพื่อทราบ และหากท่านยังไม่ได้ชำระค่าลงทะเบียน ขอความกรุณาชำระค่าลงทะเบียนรายละเอียดตามเอกสารสิ่งที่ส่งมาด้วย 2. ทั้งนี้ หากมีข้อสงสัยประการใดสอบถามได้ที่ *คณะอนุกรรมการฝ่ายวิชาการ การจัดเสนอผลงานวิจัยฯ มหาวิทยาลัยเทคโนโลยีสุรนารี ถ. มหาวิทยาลัย ต. สุรนารี อ. เมือง จ. นครราชสีมา 30000 โทรศัพท์ : 0-4422-4033-4 โทรสาร : 0-4422-4033* หรือดูรายละเอียดเพิ่มเติมได้ที่ Web Site : http:// www.sut.ac.th/gradresearch3

อนึ่ง คณะอนุกรรมการฝ่ายวิชาการได้ส่ง 1) รายละเอียดการนำเสนอผลงานวิจัยแบบโปสเตอร์แบบบรรยาย 2) กำหนดการประชุม 3) แผนผังมหาวิทยาลัยเทคโนโลยีสุรนารี มาด้วยแล้ว

ขอแสดงความนับถือ

(รองศาสตราจารย์ ดร. กฤษณะ สาคริก)
รองอธิการบดีฝ่ายวางแผน
ประธานคณะอนุกรรมการฝ่ายวิชาการ
การประชุมเสนอผลงานวิจัยระดับบัณฑิตศึกษาของประเทศไทย ครั้งที่ 3

คณะอนุกรรมการฝ่ายวิชาการฯ
โทรศัพท์ : 0-4422-4033-4
โทรสาร : 0-4422-4033
111 ถนนมหาวิทยาลัย ตำบลสุรนารี อำเภอเมือง จังหวัดนครราชสีมา 30000 โทรศัพท์ (044) 223000 โทรสาร (044) 224070
SURANAREE UNIVERSITY OF TECHNOLOGY
111 UNIVERSITY AVENUE, SUB DISTRICT SURANAREE, MUANG DISTRICT, NAKHON RATCHASIMA 30000, THAILAND Tel. (044) 223000 Fax. (044) 224070

ชื่อบทความ	การศึกษาผลกระทบของขนาดคิวของตัวจัดการแบนด์วิดท์ต่อปริมาณความแออัดของข้อมูลที่เกิดขึ้นบนระบบเครือข่าย The Effect of Queue Sizes of a Bandwidth Manager to Network Congestion
กลุ่มสาขาวิจัย	วิศวกรรมศาสตร์
ผู้นำเสนอบทความ	หฤทัย สมบูรณ์รุ่งโรจน์
สถาบันการศึกษา	มหาวิทยาลัยสงขลานครินทร์

บทนำ

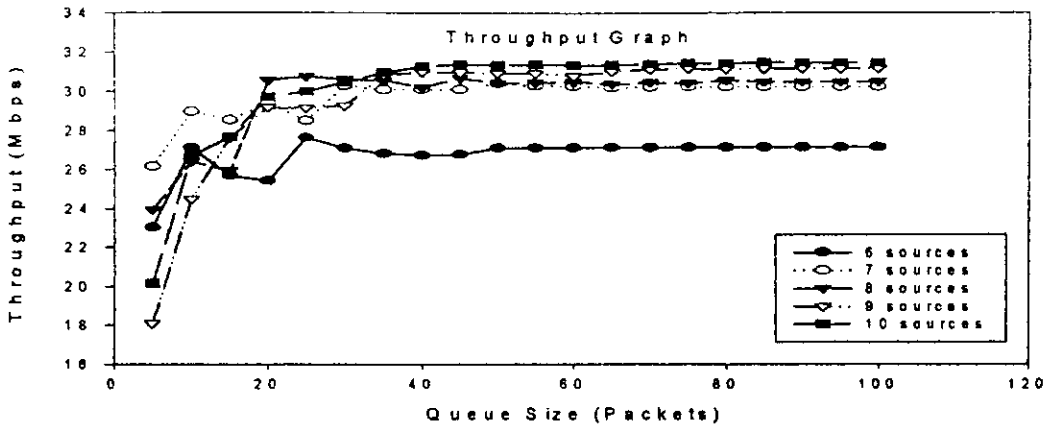
ปัจจุบันระบบเครือข่ายคอมพิวเตอร์ทั้งอินเตอร์เน็ตและอินทราเน็ตมีการขยายตัวอย่างรวดเร็ว ทำให้ระบบเครือข่ายมีขนาดใหญ่และซับซ้อนมากขึ้น รวมทั้งจำนวนเครื่องลูกข่ายก็เพิ่มสูงขึ้นด้วย ดังนั้นเพื่อให้สามารถใช้แบนด์วิดท์ (Bandwidth) ที่มีอยู่อย่างจำกัดร่วมกันได้อย่างมีประสิทธิภาพสูงสุด จึงต้องมีการจัดการข้อมูลที่เกิดขึ้นบนระบบเครือข่าย (Traffic Management) โดยใช้วิธีการจัดคิว (Queuing) ในบทความนี้ใช้อัลกอริทึมในการจัดคิวแบบจัดตามลำดับความสำคัญ (Priority Queuing) โดยข้อมูลที่ได้รับค่าลำดับความสำคัญสูงกว่า จะได้สิทธิ์ในการใช้ช่องสื่อสารก่อน ส่วนข้อมูลที่ได้รับค่าลำดับความสำคัญต่ำกว่าจะถูกพักไว้ในคิวและรอจนกว่าช่องสื่อสารจะว่าง ซึ่งจะพบว่าถ้าคิวมีขนาดเล็ก ทำให้เกิดการสูญหายของข้อมูลเป็นจำนวนมากเนื่องจากข้อมูลล้นคิว แต่ในการเพิ่มขนาดของคิวหมายถึงการเสียค่าใช้จ่ายเพิ่มมากขึ้น และยังส่งผลให้ความแออัดที่เกิดขึ้นบนระบบเครือข่ายสูงขึ้น ทั้งนี้เนื่องจากผู้ส่งข้อมูลจะกำหนดเวลาในการรอคอยการตอบรับ (Acknowledgement) ของข้อมูลจากผู้รับข้อมูล หรือที่เรียกว่า Time out เมื่อหมดเวลาดังกล่าว ผู้ส่งจะตัดสินใจทำการส่งข้อมูลชุดเดิมเข้าไปในระบบเครือข่ายใหม่ เนื่องจากคิดว่าข้อมูลชุดแรกที่ส่งไปเกิดการสูญหาย ทั้งที่จริงแล้วข้อมูลยังคงค้างอยู่ในคิว การกระทำเช่นนี้จึงเป็นการเพิ่มปริมาณความแออัดให้กับระบบโดยไม่จำเป็น ในการศึกษาผลกระทบของขนาดคิวของตัวจัดการแบนด์วิดท์ที่มีต่อปริมาณความแออัดของข้อมูลที่เกิดขึ้นบนระบบเครือข่าย เป็นเรื่องยากที่จะทำการทดลองโดยการแก้ไขหรือปรับปรุงในระบบที่ใช้งานอยู่ เพราะอาจส่งผลกระทบต่อระบบเครือข่ายทั้งระบบ ตัวจำลองระบบเครือข่าย (Network Simulator) ได้เข้ามามีบทบาทสำคัญและเป็นอีกทางเลือกหนึ่งที่ใช้เป็นเครื่องช่วยในการศึกษาและทดลองสมมติฐานต่างๆ เนื่องจากมีข้อดีหลายประการ อาทิเช่น มีค่าใช้จ่ายที่ต่ำและช่วยประหยัดระยะเวลาในการทดลอง สามารถศึกษาปัญหาที่ซับซ้อนผ่านเงื่อนไขต่างๆ ที่กำหนดได้ง่ายและรวดเร็ว รวมทั้งมีความยืดหยุ่นในการปรับเปลี่ยนและทำซ้ำเพื่อหาผลการทดลองที่ดีที่สุด

ระเบียบวิธีวิจัยที่ใช้หรืออุปกรณ์และวิธีการ

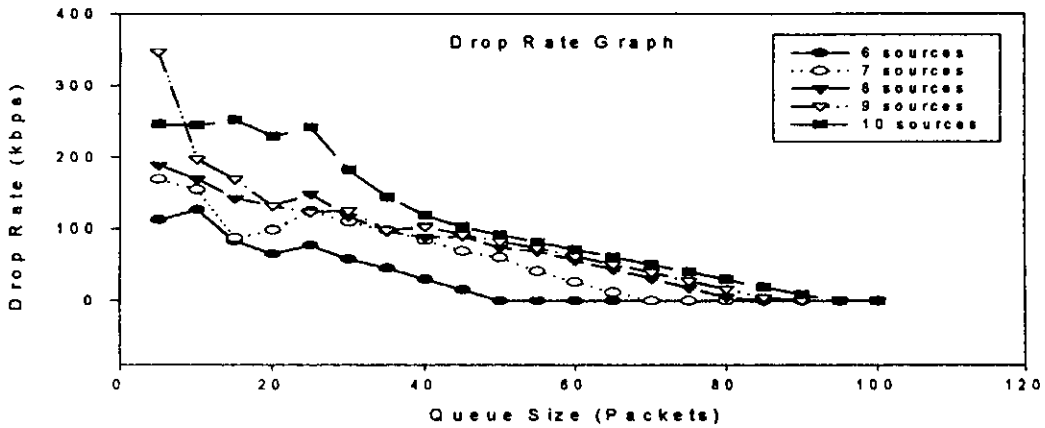
ในที่นี้เลือกทำการทดลองบนตัวจำลองระบบเครือข่ายชื่อ NS 2.0 ทำงานบนระบบปฏิบัติการยูนิกส์ โดยการสร้างแบบจำลองโทโปโลยีของระบบเครือข่ายคอมพิวเตอร์มหาวิทยาลัยสงขลานครินทร์วิทยาเขตหาดใหญ่ พร้อมทั้งทำการติดตั้งตัวจัดการแบนด์วิดท์ซึ่งใช้วิธีการจัดคิวแบบจัดตามลำดับความสำคัญไว้ตรงจุดที่เกิดปัญหาคอขวด ทำการทดลองโดยให้แหล่งข้อมูลต้นทางทำการถ่ายโอนข้อมูล (File Transfer) ไปยังแหล่งข้อมูลปลายทางผ่านทางตัวจัดการแบนด์วิดท์ และทำการปรับขนาดคิวของตัวจัดการแบนด์วิดท์จากเล็กไปจนกระทั่งใหญ่มากๆ บันทึกค่าอัตราการรับส่งข้อมูล (Throughput) ที่เกิดขึ้น รวมทั้งค่าอัตราการสูญหายของข้อมูล (Drop Rate) ด้วย

ผลการวิจัย

ผลการวิจัยแสดงดังรูปที่ 1. และ 2. พบว่าเมื่อคิวมีขนาดเล็กอัตราการสูญหายของข้อมูลมีค่าสูง ส่วนอัตราการรับส่งข้อมูลมีค่าต่ำ การเพิ่มขนาดของคิวมีผลทำให้ค่าอัตราการรับส่งข้อมูลสูงขึ้น แต่เมื่อคิวมีขนาดใหญ่จนถึงจุดๆ หนึ่ง การเพิ่มขนาดของคิวจะไม่ส่งผลต่ออัตราการรับส่งข้อมูลอีกต่อไป โดยขนาดคิวควรมีค่าอย่างน้อยประมาณ 9-10 เท่าของจำนวนแหล่งข้อมูล (Sources) เพื่อไม่ให้เกิดการสูญหายของข้อมูล ยกตัวอย่างเช่น ถ้ามีจำนวนแหล่งข้อมูล 6 แหล่ง ทำการถ่ายโอนข้อมูลไปยังปลายทาง โดยขนาดแพ็กเกตเท่ากับ 1,460 ไบต์ ดังนั้นคิวควรมีขนาดเท่ากับ 9x6 เท่ากับ 54 แพ็กเกต หรือเท่ากับ 54x1,460 ประมาณ 80 กิโลไบต์นั่นเอง ซึ่งค่าที่ได้จะนำไปใช้ในการออกแบบขนาดของบัฟเฟอร์ของตัวจัดการแบนด์วิดท์



รูปที่ 1. แสดงความสัมพันธ์ระหว่างอัตราการรับส่งข้อมูล (เมกกะบิตต่อวินาที) กับขนาดคิว (แพ็กเกต)



รูปที่ 2. แสดงความสัมพันธ์ระหว่างอัตราการสูญหายของข้อมูล (กิโลบิตต่อวินาที) กับขนาดคิว (แพ็กเกต)

อภิปรายผลและข้อเสนอแนะ

บทความนี้เสนอผลการศึกษาและวิเคราะห์ผลกระทบของขนาดคิวของตัวจัดการแบนด์วิดท์ต่อปริมาณความแออัดของข้อมูลที่เกิดขึ้นบนระบบเครือข่าย ซึ่งจากผลการวิจัยพบว่าในขณะที่คิวมีขนาดเล็กเกินไป ข้อมูลเกิดการสูญหายเป็นจำนวนมากเนื่องจากข้อมูลล้นคิว ทำให้ผู้ส่งต้องทำการส่งข้อมูลชุดเดิมเข้าไปในระบบเครือข่าย และเป็นเหตุให้ปริมาณความแออัดของข้อมูลที่เกิดขึ้นบนระบบเครือข่ายเพิ่มมากขึ้น และถ้าคิวมีขนาดใหญ่ขึ้น ค่าอัตราการสูญหายของข้อมูลมีค่าต่ำลง และทำให้ปริมาณความแออัดของข้อมูลที่เกิดขึ้นบนระบบเครือข่ายลดลง แต่เมื่อคิวมีขนาดเพิ่มขึ้นจนถึงจุดหนึ่ง การเพิ่มขนาดของคิวต่อไปจะไม่ส่งผลกระทบต่ออัตราการรับส่งข้อมูลอีกต่อไป และพบว่าขนาดคิวที่เหมาะสมขึ้นอยู่กับหลายๆ ปัจจัย ไม่ว่าจะเป็นขนาดของแพ็กเกต แบนด์วิดท์ของลิงค์ (Link Bandwidth) จำนวนแหล่งข้อมูล ประเภทของโปรแกรมประยุกต์ที่ใช้ รวมถึงอัลกอริทึมที่ใช้ในการจัดคิว

เอกสารอ้างอิง

[1] Network Simulator 2.0 (Ns2.1b8a) [Online]. Available at http://www.isi.edu/nsnam/ns/.

[2] Lawrence S. Bralmo, Larry L. Peterson. 1996. "Experiences with Network Simulation", University of Arizona.

คำสำคัญ ขนาดคิว ตัวจัดการแบนด์วิดท์ ตัวจำลองระบบเครือข่าย

ที่อยู่ที่สามารถติดต่อได้ ภาควิชาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์ มหาวิทยาลัยสงขลานครินทร์

อีเมล์ haruthai@ratree.psu.ac.th หรือ noonoi_jai@hotmail.com โทรศัพท์ (074) 212894 โทรสาร (074) 459395