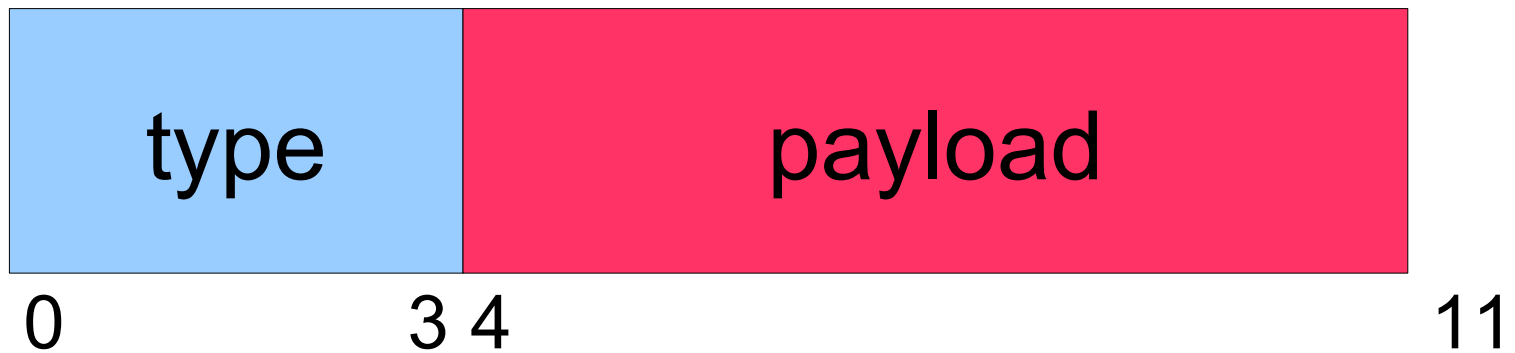


Создание сетевого протокола в **ns-2**

мастер-класс

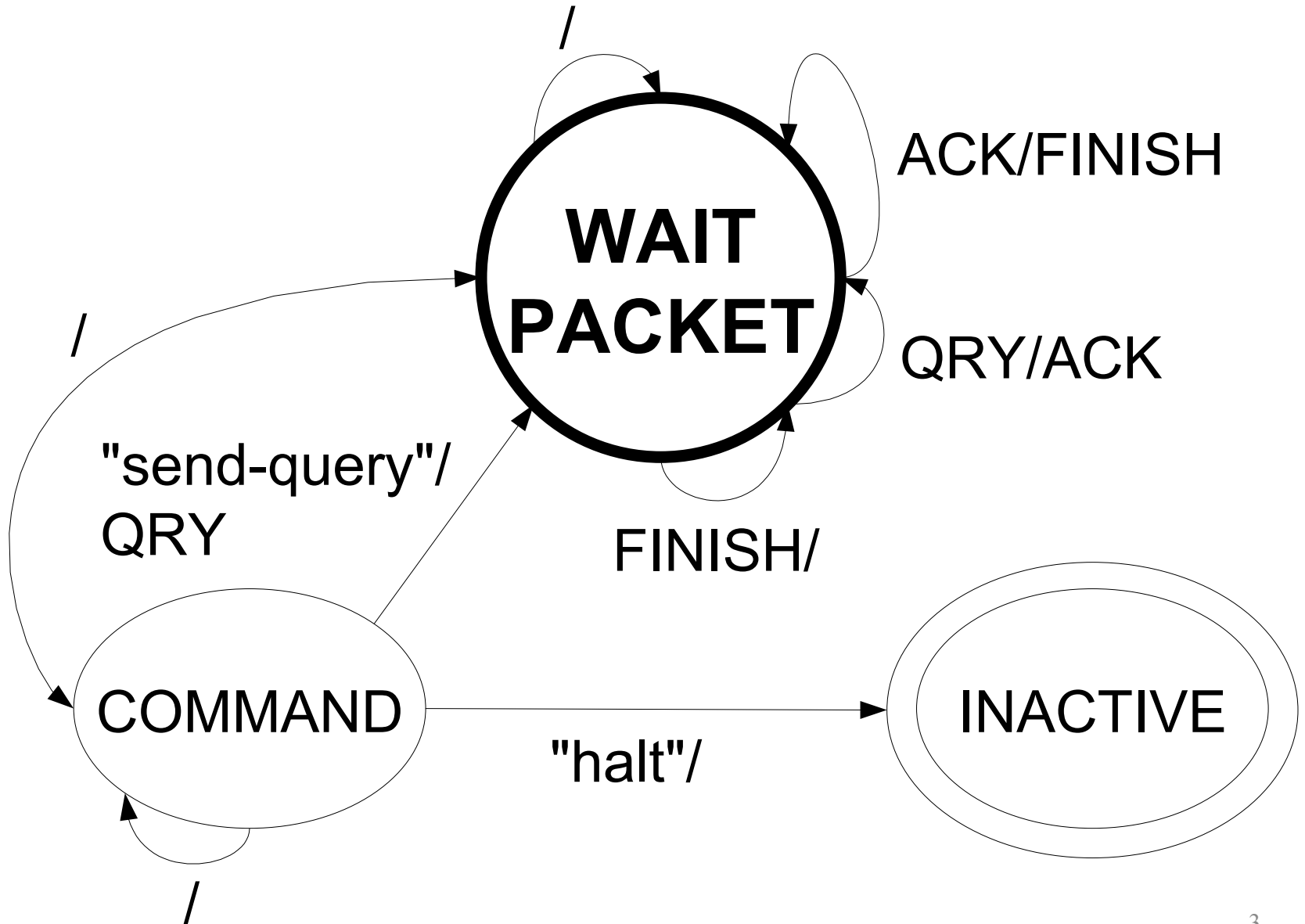
Разрабатываемый протокол

- YOP (Your Own Protocol)
- Пакет протокола имеет 2 поля:



- *type* – тип пакета (QRY, ACK, FINISH)
- *payload* – передаваемые данные
- Протокол базируется на транспорте UDP

Разрабатываемый протокол



Особенности симулятора ns-2

- Два языка программирования
 - C++
 - Все, что требует быстрой обработки сетевых пакетов или побитового доступа к содержимому пакетов
 - Tcl
 - Маршрутизация
 - Задание и изменение параметров модели
 - Процедуры для настройки протокола на узлах

Особенности симулятора ns-2

- Код на OTcl

- Реализация сетевого протокола и вспомогательные классы

- **составная часть** исполняемого файла симулятора
 - для изменения — **перекомпиляция** симулятора
 - путь к этому коду включается в Makefile

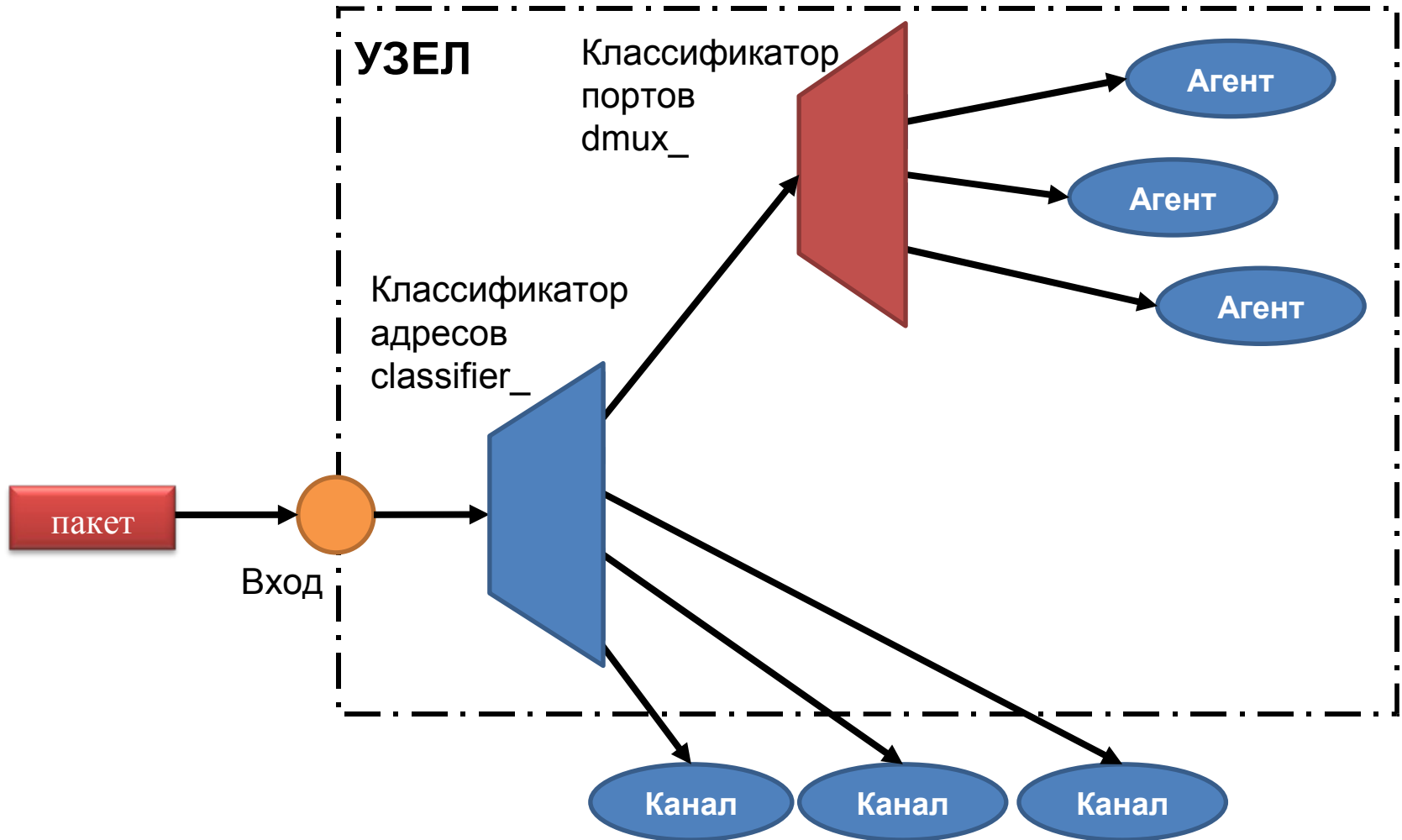
- Построение и параметризация модели

- существует **отдельно** от симулятора
 - после изменения требуется **повторно запустить** модель в симуляторе

Структура каталогов

- Код на C++
 - *~/ws/ns-allinone-2.xx/ns-2.xx/имя_протокола*
 - Принято:
 - Заголовки и код для связи с Tcl — *.h*
 - Основной код — *.cc*
- Код протокола на Tcl
 - *~/ws/ns-allinone-2.xx/ns-2.xx/tcl/имя_протокола*
- Все пути к файлам будем задавать **ОТНОСИТЕЛЬНО** *~/ws/ns-allinone-2.xx/ns-2.xx*

Обработка сетевых пакетов



Пакет

- В ns-2 пакет любого сетевого протокола моделируется классом **Packet**.
- Когда пакет принимается или отправляется, идет передача указателя на экземпляр **Packet**.
- Создание экземпляра: `Packet *p = allocpkt();`
- Уничтожение экземпляра: `Packet::free(p)`
- Пакет содержит **заголовки**, относящиеся к различным протоколам. Всегда есть **общий заголовок** (`hdr_cmn`)

Заголовок пакета

- **Заголовок пакета** — структура с именем `hdr_<имя_протокола>`
- Структура `hdr_<имя_протокола>` должна обязательно содержать:
 - Статическое поле `offset_` — смещение, с которого начинается заголовок нашего протокола
 - Статический метод `access()`, возвращающий указатель на первый байт заголовка пакета

```
/** @file yop/yop_packet.h */
struct hdr_yop {
    // поля протокола
    int32_t type;
    int64_t payload;
    // заголовок пакета
    static int offset_;
    inline static int& offset()
        { return offset_; }
    inline static hdr_yop*
    access(Packet *p) {
        return (hdr_yop*)
            p->access(offset_);
    }
};
```

Заголовок пакета

- Также желательно объявить макрос HDR_<имя_протокола>, делающий то же самое, что и access()
- Для связи с Tcl, создается класс <имя_протокола>HeaderClass. Он указывает, какой **размер** и **смещение** имеет пакет в общей структуре Packet.

NB! bind_offset, bind_bool, bind связывают адрес переменной в C++ с ее именем в Tcl коде

```
/** @file yop/yop_packet.h */
#define HDR_YOP(p)
    ((hdr_yop*)
    (p)->access(hdr_yop::offset_))

/* Packet Header for Tcl */
int hdr_yop::offset_;
static class YOPHeaderClass :
    public PacketHeaderClass {
public:
    YOPHeaderClass() :
        PacketHeaderClass("PacketHeader/YOP", sizeof(hdr_yop))
    {bind_offset(&hdr_yop::offset_;);}
} class_yophdr;
```

Тип пакета (packet.h)

Класс *Packet* хранит тип пакета в поле *ptype* общего заголовка *hdr_cmn*. Чтобы работал наш протокол, надо модифицировать объявления класса *Packet*:

- Добавить новый тип пакета, увеличив номер `PT_NTTYPE` на 1
- Добавить имя пакета в допустимые имена пакетов (метод `initName()` класса `p_info`)

```
/** @file common/packet.h
static const packet_t PT_YOP
    = 63;
// insert new packet types here
static const packet_t PT_NTTYPE
    = 64;
    // ...
class p_info {
    // ...
    static void initName() {
        // ... Many entries
        name_[PT_YOP] = "YOP";
        // ...
    }
};
```

Отсл. Заголовки пакета

- По умолчанию, ns-2 включает в пакет **все заголовки зарегистрированных в симуляторе протоколов**
- Чтобы наш заголовок был включен, его надо прописать в *tcl/lib/ns-packet.tcl*.

```
# /** @file tcl/lib/ns-  
    packet.tcl
```

```
foreach prot {  
    // ...  
    YOP  
    // ...  
}
```

```
# При этом должен  
    существовать класс  
    PacketHeader/YOP
```

Агент

- **Агент** — это сущность, обрабатывающая пакеты заданного сетевого протокола на определенном узле.
- Агент инстанцируется из Tcl и связывается с узлом:

```
set tcpAgent [new Agent/TCPAgent]  
$node attach $tcpAgent $port
```
- Агенты, осуществляющие передачу данных от одного узла к другому, соединяются друг с другом с помощью метода connect

```
$ns connect $source $sink
```

где \$ns – экземпляр класса Simulator

Агент

- Класс агента наследует от **Agent**
- Дополнительные действия по инициализации агента обычно выносят в процедуру инициализации в файле *tcl/имя_протокола/имя_протокола.tcl*, и требуют, чтобы вместо `attach` была запущена именно она
 - Такая процедура должна возвращать экземпляр созданного агента

Используемые функции

- Обработка полученного пакета

void Agent::recv(Packet *, Handler *)

- Посылка ответа

Agent::send(Packet *, Handler*)

- Исполнение команд из Tcl

Agent::command(**int** argc, **const char *const** *argv[])

– argc — количество аргументов в argv[]

– argv[0] — "cmd" (всегда)

– argv[1] — название команды

– argv[2..(argc-1)] — аргументы команды (если есть)

Агент YOP

- Агент будет двухсторонним (посылка данных от источника к приемнику) — так проще
 - Будем принимать пакеты с помощью `recv`
 - Будем отправлять ответы с помощью `send`
- Агент будет поддерживать несколько команд:
 - Состояние **COMMAND** конечного автомата протокола:
 - `send-query <payload>` — послать `YopQuery` с `payload = <payload>`
 - `halt` — прекратить обработку YOP-пакетов

Агент YOP

- Наш агент будет поддерживать несколько команд:
 - Конфигурирование
 - `bogus-config` — выводит сообщение о загрузке конфигурации по умолчанию
 - `bogus-set <flagname>` — выводит сообщение, что опция `<flagname>` установлена

Таймер

- После отправки `YopQuery` («запрос») клиент ждет ответа `YopAsk` («подтверждение») в течение 0.5 с
- Для ожидания в ns-2 есть класс `TimerHandler`
 - `sched(t)`: запустить таймер, срок ожидания *t*.
Срабатывает только, если таймер еще не запущен
Практически не используется
 - `resched(t)`: запустить или перезапустить таймер, срок ожидания *t*
 - `cancel()`: завершить ожидание до истечения срока
 - `expire()` **pure virtual**: вызывается, когда срок ожидания истек

Код агента: включаемые файлы

```
/**@file yop.h */
```

```
// класс «Пакет»
```

```
#include "packet.h"
```

```
// класс «Агент»
```

```
#include "agent.h"
```

```
// класс «Обработчик таймера»
```

```
#include "queue.h"
```

```
// структура «Заголовок YOP пакета»
```

```
#include "yop_packet.h"
```

Код агента: таймер

```
/** @file yop.h */  
typedef void (YOPAgent::*timer_fn)();  
  
class YOPTmrHandler: public TimerHandler {  
public:  
    static const double TIMEOUT = 0.5;  
  
    YOPTmrHandler(YOPAgent *pAgent,  
timer_fn eventFunc);  
    void expire(Event *);  
private:  
    YOPAgent *agent_;  
    timer_fn eventFunc_;  
};
```

Код агента: таймер

```
/** @file yop.cc
```

```
YOPTmrHandler::YOPTmrHandler(YOPAgent *pAgent,  
    timer_fn pEventFunc):  
    TimerHandler(), agent_(pAgent), eventFunc_(pEventFunc)  
{  
}
```

```
void YOPTmrHandler::expire(Event *e)  
{  
    (agent_ -> *eventFunc_)(e);  
}
```

Код агента: объявление

```
/** @file yop.h */  
class YOPAgent: public Agent  
{  
public:  
    YOPAgent();  
    void recv(Packet *p, Handler *h);  
    int command(int argc, const char* const *argv);  
private:  
    YOPTmrHandler ackTimer_  
    bool halted_  
    Packet* createYopPkt(yop_packet_type, yop_payload);  
    void endWaitAck();  
};
```

Код агента: конструктор

```
/** @file yop.cc */
```

```
YOPAgent::YOPAgent():
```

```
    // Указываем, что транспорт UDP
```

```
    Agent(PT_UDP),
```

```
    // При истечении АСК-таймера вызывается
```

```
    // функция-член endWaitAck
```

```
    ackTimer_(this, &YOPAgent::endWaitAck),
```

```
    // По умолчанию YOP-пакеты обрабатываются
```

```
    halted_(false)
```

```
{  
}
```

Код агента: recv()

```
/** @file yop.cc */  
void YOPAgent::recv(Packet *p, Handler *h)  
    if ( halted_ ) { Packet::free(p); return; }  
    hdr_yop *rcvHeader = HDR_YOP(p);  
    switch ( rcvHeader->type() ) {  
        case YopQuery:  
            Packet *ack = createYopPkt( . . . );  
            // . . .  
            send(ack, 0); break;  
        case . . .  
        default: assert("This should never happen");  
    }  
    Packet::free(p);  
}
```

Код агента: command()

```
/** @file yop.cc */  
int YOPAgent::command(int argc, const char* const *argv) {  
    if ( argc == 2 ){  
        if ( strcmp(argv[1], "bogus-config") == 0 ) {  
            ...  
            return TCL_OK;  
        }  
        else if . . . { }  
    }  
    else if . . . { }  
  
    // No match. Command is not YOPAgent-specific  
    return Agent::command(argc, argv);  
}
```

Код агента: createYopPkt()

```
/** @file yop.cc */  
Packet *YOPAgent::createYopPkt(int32_t type, int64_t payload) {  
    Packet *pkt = allocpkt();  
    // Задаем поле «протокол» в общем заголовке  
    // Иначе в файле трассы в графе «протокол» будет  
    // стоять «udp»  
    hdr_cmn *hdrCmn = HDR_CMN(pkt);  
    hdrCmn->ptype_ = PT_YOP;  
  
    hdr_yop *hdrYop = HDR_YOP(pkt);  
    hdrYop->type_ = type;  
    hdrYop->payload_ = payload;  
    return pkt;  
}
```

Код агента: обертка для Tcl

```
/** @file yop.h  
static class YOPAgentClass: public TclClass {  
public:  
YOPAgentClass(): TclClass("Agent/YOP") { }  
  
TclObject *create(int, const char* const *) {  
    return new YOPAgent();  
}  
} class _yopagent;
```

Tcl. Привязка к узлу

- Добавляем метод `start-yop` к методам класса «узел» (`Node`), чтобы можно было запустить YOP на узле, написав

```
$node start-yop
```

- В нашем простом случае этот метод сводится к вызову `attach`:

```
Node instproc start-yop {} {  
  set default_port 8555 ;#const  
  set yop [new Agent/YOP]  
  $self attach $yop $default_port  
  return $yop  
}
```

Добавления в *Makefile.in*

- Добавить ссылки на C++ в *Makefile.in*
 - на исходный код агентов и классификаторов:

```
INCLUDES = \ . . .
```

```
-I./уор
```

– на объектные файлы:

```
OBJ_CC = \ . . .
```

```
уор/уор.o \
```

Добавления в *Makefile.in*

- Добавить ссылку на файл со вспомогательными процедурами в *tcl/lib/ns-lib.tcl*

```
# PMIPv6
```

```
source ../yop/yop.tcl
```

```
# end PMIPv6
```

- Добавить ссылку на файл со вспомогательными процедурами в *Makefile.in*:

```
NS_TCL_LIB = ...
```

```
tcl/yop/yop.tcl \
```

```
@V_NS_TCL_LIB_STL@
```

Задание

- Дописать недостающий код так, чтобы модель из файла *uor_example.tcl* заработала
 - Можно подглядывать в репозиторий: *ns-2/trunk/offload/master-class*, но это неспортивно
- Добавить любой параметр к протоколу и связать его с Tcl с помощью `bind`, `bind_bw` или `bind_bool`
- Добавить любую команду, изменяющую значение созданного вами параметра

Дополнительная информация

- Implementing ns-2 Protocol

<http://masimum.dif.um.es/nsrt-howto/pdf/nsrt-howto.pdf>

- ns-2 Mark Greis' tutorial

www.isi.edu/nsnam/ns/tutorial/

- Excerpts from book T. Issariyakul , and E. Hossain , Introduction to Network Simulator NS2

<http://www.ece.ubc.ca/~teerawat/NS2.htm>

- ns-2 Manual

<http://www.isi.edu/nsnam/ns/tutorial/nsdoc.html>

- ns-2 Wiki

<http://nsnam.isi.edu/nsnam/index.php/>