University of Ljubljana
Faculty of Computer and Information Science

Modeliranje računalniških omrežij

# OMNeT++
# Kanali in paketi

Laboratorijske vaje

Torek, 5. novembra 2013

# Kanali (Channels)

- z njimi določimo lastnosti povezav
- v ozadju so C++ razredi (kot pri *simple* modulih) in NED datoteke
- ponavadi ni potrebno pisati svojih kanalov, ampak izhajamo iz predefiniranih
  - `ned.IdealChannel`
  - `ned.DelayChannel`
  - `ned.DatarateChannel`
- vključimo `import ned.*`

# Tipi kanalov

`IdealChannel`

- brez parametrov
- simulira idealni kanal
- privzeti kanal

`DelayChannel`

- `delay` (`double`): zakasnitev v enotah (`s`, `ms`, `us`, itd.)
- `disabled` (`bool`): `false` – kanal bo zavrgel vse pakete

# Tipi kanalov

`DatarateChannel`

- `delay` (`double`), `disabled` (`bool`): zakasnitev pri pošiljanju
- `datarate` (`double`):
  - hitrost prenosa (`Kbps`, `Mbps`, itd.)
  - upošteva se pri računanju časa prenosa
  - 0 = neomejena hitrost (default)
- `ber` in `per` (`double, [0,1]`):
  - Bit/Packet Error Rate
  - za modeliranje napak
  - nastavi `Error flag` v paketu

# Specificiranje kanalov (NED)

- specificiramo jih v NED datoteki, ki se nanaša na omrežje ali sestavljen modul
- v `types` delu NED datoteke specificiramo kanal

```
channel C1 extends ned.DatarateChannel
{
   datarate = 100Mbps;
   delay = 100us;
   ber = 1e-10;
}
```

- kanalu lahko dodajamo svoje parametre

```
channel C2 extends ned.DatarateChannel
{
   double distance @unit(m);
   delay = this.distance / 200km * 1s;
}
```

# Uporaba kanalov

- v `connections` delu NED datoteke

```
ime_modula1.ime_vrat <--> ime_kanala <-->
                      --> ime kanala -->
            <--  ime kanala <--
ime_modula2.ime_vrat
```

- primer

```
source.out --> C1 --> queue.in++;
queue.out --> C2 --> sink.in;
```

# Opisovanje kanalov (C++)

- izhajamo iz razreda `cChannel` ali katere njegove izpeljave
- makro `Define_Channel()` – uporaba kot `Define_Module()`
- implementirati moramo 3 metode:
- `bool isTransmissionChannel()`: ali je prenosni kanal – ali modeliramo zakasnitev prenosa – kliče setDuration() cPacket-a
- `simtime_t getTransmissionFinishTime()`: samo pri prenosnih kanalih – kdaj bo/je bil prenos končan – kličejo moduli
- `void processMessage(cMessage *msg, simtime_t t, result_t& result)`: implementira funckionalnosti kanala
  - spreminjamo strukturo `result_t`:

```
struct result_t {
 simtime_t delay; // propagation delay
 simtime_t duration; // transmission duration
 bool discard; // whether the channel has lost the message
 };
```

  - transmission duration in modeliranje napak vplivata le na pakete (`cPacket`)

# Paketi (Packets)

- `cPacket` se uporablja za modeliranje omrežnih paketov v komunikacijskih omrežjih (izpeljava `cMessage`)
- modeliranje zakasnitve prenosa in napak pri prenosu vplivata le na pakete (na sporočila nimata vpliva)
- lastnosti:
  - `packet length`: v bitih – za izračun trajanja prenosa (*transmission duration*) in izračun modeliranja napak pri prenosu

  (nastavljanje z metodami `setByteLength` in `setBitLength` )
  - `bit error flag`: ali je prišlo do napake; sprejemnik opcijsko preverja ta bit
  - `duration`: sem se shrani trajanje prenosa po prenosu

```
cPacket *msg = new cPacket(name, kind, bitLegnth);
```

vsi parametri so opcijski

# Kanali (Channels)

- kanali so vezani na vrata (*gates)*

```
cChannel *channel = gate->getChannel();
cGate *gate = channel->getSourceGate();
```

- ugotavljanje ali je kanal "transmission channel" (modelira "transmission duration")

```
channel->isTransmissionChannel()
```

- dostopanje do prenosnega kanala vrat

```
cChannel *channel =
   gate->getTransmissionChannel();
```

# Pošiljanje paketa

- pošiljanje paketa: do ciljnega modula potuje skozi serijo povezav *(connection path)*
- vsaka povezava *(connection)* ima lahko svoj kanal
- samo en kanal v seriji ima lahko neničeln *transmission duration* (je tipa *DatarateChannel*)
- to je *Transmission Channel*
- paketi se lahko prenašajo, le ko je *Transmission Channel* prazen
- ali je kanal prost: `isBusy()`, `getTransmissionFinishTime()`

# Pošiljanje paketa

1. preverimo, če je prenosni kanal prost (`isBusy()` )
2. če je prost, paket takoj pošljemo
3. če ni prost, paket shranimo v čakalno vrsto in z uporabo metode `scheduleAt()` ob času `getTransmissionFinishTime()` spet preverimo, če je oddajni kanal prost

Glej modul `transmitter`.

# Sprejemanje paketa

1. preverimo, če je prišlo do napake: `hasBitError()`
2. V primeru napake paket zavržemo

Paket je dostavljen v ciljni modul, ko je prenešen v celoti. Glej modul `receiver`.

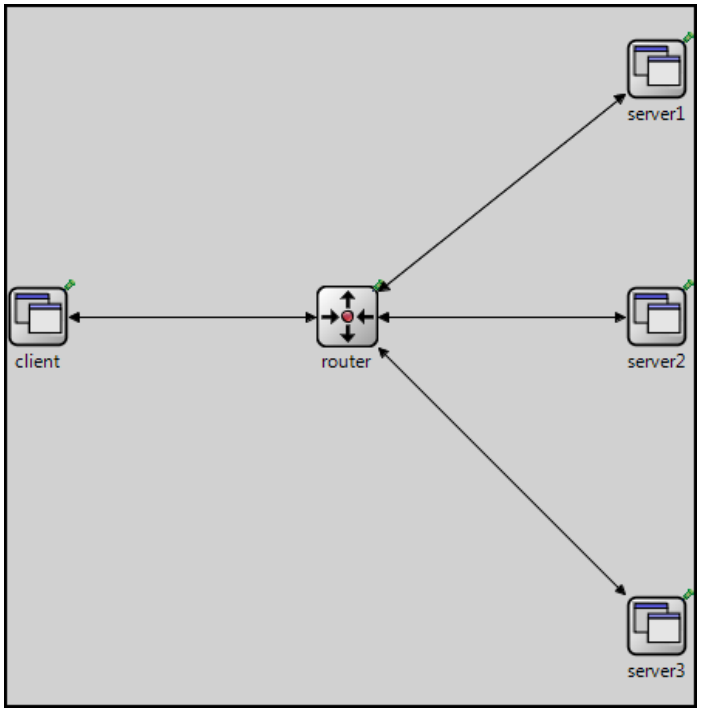
# Zgled vaja04

Zgled vsebuje sledeče *simple* module

- `mySource`: generator paketov, ki predstavljajo zahteve,
- `mySink`: ponor paketov
- `transmitter`: oddajni modul, ki oddaja samo takat, ko je prenosni kanal prost
- `receiver`: sprejemni modul, ki preveri ali je pri prenosu prišlo do napake
- `myQueue`: M/M/c sistem za simuliranje strežbe
- `responseGen`: generira odgovore na zahteve (spremeni velikost paketa)

# Naloga

- Realizirajte topologijo na sliki.



# Naloga

- `Client` = odjemalec, ki pošilja zahteve in prejema odgovore od strežnikov
- `Router` = usmerjevalnik, ki pošilja pakete na izhodne povezave glede na njihov naslov
- `Server 1,2,3` = strežniki, ki odgovarjajo na odjemalčeve zahteve

# Naloga

- odjemalec generira zahteve velikosti od 12 do 64 bajtov, pri čemer medprihodni časi nihajo od 50 do 500 ms
- 10% zahtev je namenjenih strežniku 1, 50% strežniku 2, 40% pa strežniku 3
- strežniki generirajo odgovore sledečih velikosti
  - strežnik 1: 2048 bajtov
  - strežnik 2: 512 bajtov
  - strežnik 3: 64 bajtov
- časi strežbe so sledeči
  - strežnik 1: 500 milisekund
  - strežnik 2: 100 milisekund
  - strežnik 3: 50 milisekund

# Naloga

- kapaciteta medpomnilnika v oddajnikih naj bo 10 paketov
- kapaciteta čakalne vrste v strežnikih naj bo 10 paketov
- kapaciteta čakalne vrste v usmerjevalniku naj bo 100 paketov
- usmerjevalnik za usmerjanje potrebuje 1 milisekundo
- verjetnost napake pri prenosu je za vse povezave enaka 2e-10, zakasnitev pa 100 us

# Naloga

1. določite optimalno število strežnih enot v posameznem strežniku
2. določite minimalno kapaciteto povezav, pri kateri ne bo prihajalo do izgubljanja paketov
   - predpostavljate lahko, da imate le dva tipa povezav: upload in download povezave
   - upload povezave vodijo od odjemalca proti strežnikom, download pa od strežnikov proti odjemalcu
3. Določite povprečni čas od takrat, ko odjemalec pošlje zahtevo do takrat, ko odjemalec prejme odgovor na to zahtevo
4. Beležite tudi število zavrženih paketov zaradi napak pri prenosu

# Modeliranje usmerjevalnika

- usmerjevalnik naj bo sestavljen iz sledečih modulov
  - sprejemni modul (`receiver`)
  - `x` oddajnih modulov (`transmitter`): `x` je število izhodnih povezav
  - `M/M/1` modul (`myQueue`): modelira zakasnitev pri usmerjanju paketov (1 milisekunda)
  - stikalo (`switch`): pošlje paket na ustrezen izhodni modul glede na naslov paketa
    - `client` = naslov 0
    - `server1` = naslov 1
    - `server2` = naslov 2
    - `server3` = naslov 3

```
send(packet,"out",packet->getAddress());
```