Advanced Network Security
1. Course Outline

Jaap-Henk Hoepman

Digital Security (DS)
Radboud University Nijmegen, the Netherlands
\textit{x@cs.ru.nl} // \texttt{www.cs.ru.nl/~jhh}

About me

About you
 Administrative details

- **Course code:** NWI3MC050, 5 ects
- **Written exam**
- **Course website**
  - Not using Blackboard
  - Instead see: http://www.cs.ru.nl/~jhh/ans.html
- **Literature**
  - Selected papers, see website.

Rough lecture setup

- **Several papers per lecture**
  - Read them in advance
  - Ask questions you may have about them after lecture
- **Homework**
  - Not graded
  - But discussed at start of next lecture
- **Feedback wanted!**
  - Because this is a new course
Course contents

Advanced Network Security

CIA
Availability
Privacy

Distributed algorithms

THE WHOLE INTERNET

Jaap Henk Hoepman
Radboud University Nijmegen
Consensus

- How could you solve it

Fault tolerance: self stabilisation
Self-stabilisation

- How could it be achieved

Bitcoin

Privacy friendly networking
Randomised algorithms

Meta-knowledge
- Distributed algorithms
  - Modelling
  - Reasoning
  - Designing: "Algorithmics"
- Global 'emergent' behaviour based on local decisions
- The 'forgotten' security properties
  - Availability
  - Privacy
- The power of randomisation

Questions? If not, let's hit the road!
Basics of distributed algorithms

Distributed algorithms are everywhere!

- Computer networks
  - Message passing
    - Routing
    - DNS
    - ...

- Multi-threaded applications
  - Shared memory / message passing
    - User interface
    - Browser loading page elements
    - Operating system
    - Services
    - Parallel processing

Concurrency: a basic example

```plaintext
i = 1, j = 0,
thread i = 2
thread j = 1
print j
```

What will be the output of this simple program?
A basic example: answer

- It depends
  - Events never take place instantaneous
  - Order of execution is not fixed; determined by scheduler
- Possible answers:
  - 0
  - 1
  - 2

A slightly more complex example

```c
1 = 1, 
2 = 0, 
thread 1 = 2; print j 
thread j = 1; print j
```

Again, what will be the output of this program?

A slightly more complex example: answer

- 2 2
- 0 2
- 1 1
- 0 1?? 
  - Depends….
Indivisibility of events

- If j=i and print j are 'indivisible'
  - Then 0 1 is not a possible output

- If j=i is a read of i followed by a write to j, or if print j is a read of j followed by writing the output to the screen
  - Then 0 1 is a possible output

What about infinite executions?

\[ i = 0, \]

\[ \text{thread while } i = 0 \]
\[ \quad \text{do} \]
\[ \quad \text{print } i \]
\[ \quad \text{thread } i = 1 \]

Possible outputs:
1
01
001
0001
00001
….

Also possible

Scheduling

- Scheduler determines next action to be executed
  - In a non-deterministic way
  - An executed action is called an event

- Note: scheduler is not a real system component
  - It just models the influence of (external) factors on which action is executed

- Fairness:
  - A scheduler is fair if an action that is continuously-enabled will always executed eventually
  - So: the protocol on the previous slide will eventually terminate
Non determinism vs randomness

<table>
<thead>
<tr>
<th>Non-deterministic</th>
<th>Random</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i = 0$; thread while $i = 0$ do print $i$ thread $i = 1$</td>
<td>$i = 0$; while $i = 0$ do $i = \text{random}(0,1)$; print $i$</td>
</tr>
</tbody>
</table>

Possible outputs:

Non-deterministic: 001, 0001
Random: 01, 0001, 00001

We cannot say anything about the thread.

Modelling a distributed system

- **Node (aka process)**
  - Executes a sequence of actions
  - Each action-execution is an event
  - Communicates with other nodes through shared memory or message passing

- **Graph** $G = (V, E)$ of nodes $V$ and edges $E$
  - $N = |V|$ the number of nodes
  - $(v,w) \in E$ if node $v$ can communicate data to $w$
  - Graph can be directed or undirected

Causality: Ordering events

- Let $\mathcal{E}$ be the set of events, and let $A \subseteq \mathcal{E}$ be events
- Define the "happened before" relation $\rightarrow$ as follows:
  - $a \rightarrow b$ if $a$ happened before $b$ in the same process, or
  - $a$ is a send event whose value is received by receive event $b$, or
  - $a$ is a write event whose value is read by read event $b$
- $a \rightarrow b$ and $b \rightarrow c$ then $a \rightarrow c$
- We assume $s \rightarrow s$
- If neither $a \rightarrow b$ nor $b \rightarrow a$ then they are concurrent $a \parallel b$
- We sometimes write $a \nrightarrow b$ to visually emphasize that $a$ may precede $b$
- $a \nrightarrow b$ means that $a$ can have a causal influence on $b$

This is a transitive partial order over all events

- Ordered by and meaning that can externally be observed
- Does not imply any "global time"
- Describing the causal order of events on each individual process
- Modelling a causal order among events
Execution

- When we run a distributed system, the nodes execute their actions. This leads to a set of events \( \mathcal{E} \).
- These events can be partially ordered using the happened before relation \( \prec \).
- This partial order can be extended to a total order \( \Rightarrow \).
- \( \mathcal{E}, \Rightarrow \) is an execution of the system.

State, configuration, evolution

- Every node has a local state.
- Every edge has also a state.
- Either the values in the shared variables or the messages in the message buffers.
- The global state of the system is the Cartesian product of all local states and edge states, and called the configuration \( \mathcal{C} \).
  - We write \( \mathcal{C}[i] \) for the state of node \( i \), and \( \mathcal{C}[j] \) variable for the value of that variable in \( i \)'s state.
  - We write \( \mathcal{E}[e] \) for the state of edge \( e \). In the shared memory model, \( \mathcal{E}[e] \) variable denotes the value of that shared variable.

Execution, evolution

- A node \( i \) may have 0 or more actions enabled.
  - Depending on its local state \( \mathcal{C}[i] \).
  - As well as the states of the incoming edges \( \mathcal{C}[j] \).
- An execution \( \mathcal{E}, \Rightarrow \) induces an evolution of the state of the system.
  - Let \( a_0, a_1, a_2, \ldots \) be the sequence of events such that \( a_i \Rightarrow a_j \) iff \( i < j \).
  - Let \( \mathcal{C}_0 \) be the initial configuration.
  - Let \( a_i \) change \( \mathcal{C}_{i-1} \) into \( \mathcal{C}_i \).
  - Then \( \mathcal{C}_0, \mathcal{C}_1, \mathcal{C}_2, \ldots \) is the evolution of the system.
Execution, evolution (continued)

- If we want to prove property \( Q \) about a distributed system, we need to prove \( Q \) for all possible executions.
- Possible executions are restricted by the ‘logic’ of the distributed system.
- This logic is captured by the happened before relation \( \rightarrow \)
- In other words: to prove property \( Q \) we need to show that for all total orders \( \Rightarrow \) that extend \( \rightarrow \) we have that \( Q \) holds for \( \langle d, \Rightarrow \rangle \)
- Such a total order is consistent with all local observations
  - Global observations (“god’s view”) do not exist.

Atomicity

- So far we have assumed events are instantaneous
  - But in reality all events take time, sometimes a long time, to complete
- For \( a, b \in A \) let \( (a) \) be the local start time and \( (b) \) be the local end time
  - Clearly \( (a) < (b) \)
  - Also, for \( a, b \in A \) from the same process, either \( (a) < (b) \) or \( (b) < (a) \)
  - The events of the same process cannot overlap
  - For \( a, b \in A \) from different processes, \( a \rightarrow b \) if \( a \) causally influences \( b \)
  - Or writing to shared memory or sending a message
  - This again fixes the partial order
- We call \( \rightarrow \) atomic
  - Atomicity is an assumption on the behaviour of actions
  - Atomic operations can be constructed using protocols using smaller building blocks.

Back to slightly more complex example

- Serialisation
  - The process of ‘shrinking’ actions to a single point in time and creating a total order.
- Assume \( j = i \) is an atomic operation
- Now assume only reads and writes are atomic
Logical clocks (Lamport paper)

- **Model**
  - Every event of a node is assigned a natural number $\pi_v^n$.
  - The logical clock of a node $v$ is defined by $\pi_v^n = \pi_{\pi_v^n} + 1$ if event $v$ occurred after $\pi_v^n$.

- **Implementation**
  - Each node maintains a counter $\pi_v$.
  - Nodes increment $\pi_v$ after every event.
  - $\pi_v$ is assigned the value of $\pi_{\pi_v}$ just before event $v$.

- **Correctness**
  - $\pi_v^n < \pi_w^n$ implies event $v$ occurred before event $w$.

- **Implementation details**
  - Each node maintains a counter $\pi_v$.
  - Initially $\pi_v = 0$ for every node.
  - Nodes increment $\pi_v$ after every event.

- **Events are now totally ordered** (by clock and node number $\pi_v^n$).

Communication

- **Shared memory**
  - Write overwrites previous value
  - Read returns last value written

- **Message passing**
  - Infinite buffers
  - Usually FIFO: First-In-First-Out
    - But messages may get dropped or reordered

Properties of distributed systems

- **Communication**
  - Shared memory vs message passing

- **Timing**
  - Synchronous, partially synchronous, asynchronous

- **Scheduling**
  - Fair, round-robin

- **Identity**
  - Uniform, random identifiers, 0..N-1

- **Network topology**
  - $G(V,E), |V| = N$
  - And whether the size of the network $N$ is known

- **Sensitivity**
  - Sense of direction
  - Run, execution
Reasoning about distributed systems

- Prove global property based on local actions
- Measure the cost of achieving this
  - Worst-case, average case
  - Time complexity: number of rounds, number of steps
  - Message complexity
  - Bit complexity
- Prove lower bounds
- Prove impossibility

Round complexity

- Round
  - The smallest period in which all nodes in the system have executed at least one action
- Any execution can uniquely split into rounds
- Round complexity
  - The number of rounds in an execution

Rationale

- Slowest node usually delays progress
- Without round complexity, asynchronous systems would have unbounded time complexity